TAMPEREEN TEKNILLINEN YLIOPISTO
TAMPERE UNIVERSITY OF TECHNOLOGY

JANNE MALM

HYPERTEKSTIN SIIRTOPROTOKOLLA 2

Diplomityö

Tarkastajat: professori Pekka Loula ja Juha Vihervaara
Tarkastajat ja aihe hyväksytty Talouden ja rakentamisen tiedekuntaneuvoston kokouksessa 4. toukokuuta 2016

TIIVISTELMÄ

Janne Malm: Hypertekstin siirtoprotokolla 2
Tampereen teknillinen yliopisto
Diplomityö, 53 sivua
Toukokuu 2016
Tietotekniikan diplomi-insinöörin tutkinto-ohjelma
Pääaine: Tietoliikennetekniikka
Tarkastajat: professori Pekka Loula ja Juha Vihervaara

Avainsanat: HTTP/2, HTTP, hyperteksti, siirtoprotokolla

Viimeisen vuosikymmenen aikana Internet-sivujen määrä, sisältö ja niiden käyttö on lisääntynyt massiivisesti. Kuitenkin niitä käytetään pääasiassa yli vuosikymmenen vanhoilla protokollilla. Verkon kaistanleveys ja tietokoneiden laskentateho on noussut huomattavasti Internetin alkuajoista. Verkon nopeus on kasvanut huomattavasti vähemmän. Verkon nopeuttamiseksi pullonkaulana olevaa viivettä palvelimien ja asiakkaan välillä pitäisi pystyä pienentämään. Viiveen pienentäminen on suuri haaste. Viivettä ei koskaan voida poistaa kokonaan, sillä signaalin kulkunopeus ja verkon topologia vaikuttavat olennaisesti viiveeseen. Verkon viiveen pienentämiseksi huomio täytyykin kääntää sovellusten rakentamiseen ja parantamaan siirtoprotokollien toimintaa. Tämän työn tarkoituksena on esitellä uusi versio käytetyimmästä protokollasta Internetin selaukseen ja antaa yleiskuva HTTP/2-protokollan toiminnasta, sekä katsauksen ensimmäisiin tutkimusaineistoihin.

Hypertekstin siirtoprotokolla (HTTP) on sovelluskerroksen protokolla, joka toimii kuljetuskerroksen protokollan TCP päällä. Sitä käytetään asiakaskoneen verkkoselaimen ja verkkosivujen palvelimien välillä siirtämään dataa. HTTP/2 on sen seuraava askel. Se parantaa toimintaa ja vähentää viivettä esittelemällä protokollaan suoraan binäärisen kehyksen, otsakkeen pakkausta, kanavointia, vuonhallintaa, priorisointia ja palvelimen työntöä. Aikaisemmissa versioissa olevia puutteita on pyritty korjaamaan palvelimien päähän tehdyillä kiertoteillä. HTTP/2 pyrkii tekemään näitä kiertoteitä turhaksi ja siirtämään toiminnallisuutta suoraan itse protokollaan.

Työssä esitellään HTTP/2:n alla toimiva TCP-protokolla, HTTP:n aikaisemmat versiot ja niiden ongelmat, sekä HTTP/2:n pohjana toiminut SPDY-protokolla. Käydään läpi HTTP/2:n rakenne tarkemmin ja esitellään neljä tutkimusta HTTP/2:n tehokkuudesta aikaisempaa HTTP/1.1:tä vastaan.

Testeissä HTTP/2 osoittaa kasvanutta suorituskykyä erityisesti verkon kiertoajan kasvaessa. HTTP/2 ei hidastu pyyntöjen määrän kasvaessa kuten HTTP/1.1 jos haettavan datan määrä ei kasva. Testit ovat osittain ristiriidassa keskenään johtuen luoduista testausolosuhteista. Laajin tutkimus on sisältää suuren näytteen Internetin yli suoritetuista latausajoista oikeilta palvelimilta. Näissä testeissä 70-85 prosenttia testatuista sivuista toimivat nopeammin HTTP/2:n kanssa. Erityisesti palvelimen työntö osoittaa hyödyntämätöntä potentiaalia.

ABSTRACT

Janne Malm: Hypertext transfer protocol 2
Tampere University of Technology
Master of Science Thesis, 53 pages
May 2016
Master's Degree Programme in Information Technology
Major: Telecommunications
Examiner: prof. Pekka Loula and Juha Vihervaara

Keywords: HTTP/2, HTTP, hypertext, transfer protocol

During the last decade Internet pages have increased in both size and usage. However the main protocols used are over a decade old. Bandwidth and the calculation power of computers have grown significantly from the early days of Internet. The speed of browsing the web however has not increased as much. The bottleneck of the web is latency between the user and the server. We should be able to reduce the latency but it will be a great challenge. Latency cannot be removed because a signal takes time to travel and the topology of the web. We must improve applications and underlying transport protocols to reduce the latency. The meaning of this work is to introduce the new version of the most widely used protocol of surfing the Internet and give you an overview of HTTP/2 with a view to the first studies.

Hypertext transfer protocol (HTTP) is a protocol for the application layer. HTTP functions on top of a transfer layer protocol TCP. HTTP is used to transfer data between client browser and a server of a web page. HTTP/2 is the next step. It improves the functionality of the protocol by introducing binary frame, header packing, multiplexing, flow control, priority and server push. Earlier versions have tried to improve their functionality with server side roundabouts. HTTP/2 tries to remove these roundabouts by moving their functionality straight in to the protocol.

This work presents the underlying TCP protocol, earlier versions of HTTP and their problems with the SPDY protocol as foundation of HTTP/2. We will go through the HTTP/2 protocol in detail and present four different studies where the performance of HTTP/2 is compared to the earlier HTTP/1.1.

In the tests HTTP/2 shows improved performance especially as the round trip time increases. Unlike HTTP/1.1, HTTP/2 does not slow down when the amount of requests increases if the amount of data stays the same. The tests are in conflict with each other because of the test conditions. A study that was ran over the Internet with real servers was considered to be the most comprehensive one. Those tests show that 70-85 percent of the pages that were tested function fasters with HTTP/2. Especially server push functionality shows potential that is unused.

SISÄLLYSLUETTELO

LYHENTEET JA MERKINNÄT

ALPN	engl. Application-Layer Protocol Negotiation Extension, TLS-salausalgoritmin laajennus sovelluskerrokselle
ASCII	engl. American Standard Code for Information Interchange, merkistö muuttamaan tietokoneen numerot merkeiksi
BDP	engl. Bandwidth-Delay Product, verkon viiveen parametri. Maksimimäärä dataa verkossa, joka on lähetetty, mutta ei vastaanotettu
CDF	engl. Cumulative Distribution Function, kuvaa mittausten sarjaa, joka kertoo mittauksen todennäköisyyden suuretta vastaan
CSS	engl. Cascading Style Sheets, tyyliohjeet erityisesti WWW-dokumenteille
DASH	engl. Dynamic Adaptive Streaming, dynaaminen mukautuva suoratoisto
DNS	engl. Domain Name System, Internetin nimipalvelujärjestelmä
DoS	engl. Denial-of-Service attack, palvelunestohyökkäys
DDoS	engl. Distributed Denial-of-Service attack, hajautettu palvelunestohyökkäys
HOL	engl. Head-of-line blocking, ensimmäisen viestin odotusongelma
HPACK	Otsakkeenpakkausalgoritmi
HTTP	engl. Hypertext transfer protocol, hypertekstin siirtoprotokolla
HTTPS	engl. Hypertext transfer protocol secure, SSL-suojattu versio HTTP:sta
IPv4	engl. Internet protocol version 4, neljäs versio Internetin protokollasta. Reitittää Internet-liikennettä.
IPv6	engl. Internet protocol version 6, kuudes versio Internetin protokollasta. Reitittää Internet-liikennettä.
JS	engl. JavaScript, kieli toiminnallisuuden lisäämiseen verkkosivulle
LTE	engl. Long Term Evolution, edistynyt 3G-tekniikka
MPD	engl. Media Presentation Description, dokumentti videotiedostosta ja niiden riippuvuuksista
NPN	engl. Next Protocol Negotiation, TLS-salausalgoritmin laajennus sovelluskerrokselle
PKI	engl. Public Key Infrastucture, tapa luottaa julkisen avaimen luotettavuuteen tietoliikenteessä
RTT	engl. Round trip time, verkon kiertoaika, eli kuinka kauan asiakkaan pyynnön lähetyksestä kestää vastauksen saamiseen
SPDY	Avoin verkkoprotokolla verkonsisällön kuljetukseen
SSL	engl. Secure Sockets Layer, tietoverkkosalausprotokolla
TCP	engl. Transmission control protocol, tietoliikenneprotokolla yhteyden luomiseen tietokoneiden välille
TLS	engl. Transport Layer Security, salausprotokolla sovellusten tietoliikenteelle
URI	engl. Uniform Resource Identifier, merkkijono, joka kertoo tiedon sijainnin
URL	engl. Uniform Resource Locator, URI:n erikoistapaus osoittamaan verkkosivuja
WS	engl. Web socket, protokolla, joka mahdollistaa kaksisuuntaisen protokollan TCP-yhteyden sisällä

WWW	engl. World Wide Web, hajautettu hypertekstijärjestelmä verkkosivujen hakemiseen

1. JOHDANTO

Yleisin tämänhetkinen tapa siirtää verkkosisältöä palvelimelta asiakkaalle on käyttää hypertekstin siirtoprotokollaa 1.1 (HTTP/1.1), joka toimii kuljetuksenhallintaprotokollan (TCP) päällä. HTTP/1.1 julkistettiin vuonna 1999 ja siitä on tullut sen jälkeen käytetyin sovelluskerroksen protokolla Internetissä. TCP on vielä vanhempi kuin HTTP/1.1, mutta on kehityksestään huolimatta olennaisilta osiltaan muuttumaton. TCP toimii Internet Protocol version 4 (IPv4) ja IPv6 päällä. (White et al. 2012)

World Wide Web (WWW) kehitettiin vuonna 1990. Sen jälkeen verkon kaistanleveys, sekä palvelimien ja asiakkaiden laskentateho on kasvanut noin puolitoistakertaiseksi vuodesta toiseen. Verkon nopeus on asiakkaan kannalta kasvanut kuitenkin paljon vähemmän. Tähän vaikuttaa verkkosivujen kasvanut monimutkaisuus. Monimutkaisuutta voidaan mitata esimerkiksi kokonaissivukoolla, linkitetyiden resurssien määrällä, palvelimen koodin määrällä ja JavaScriptin (JS) määrällä. Verkon hitauden selittäminen monimutkaisuudella on kuitenkin vain yksi mahdollisuus. Verkon alla toimivat protokollat ovat jo hyvin vanhoja ja vanhenevat koko ajan. Huomattavia parannuksia verkon suorituskykyyn voidaan saavuttaa muokkaamalla näitä perusprotokollia. (White et al. 2012)

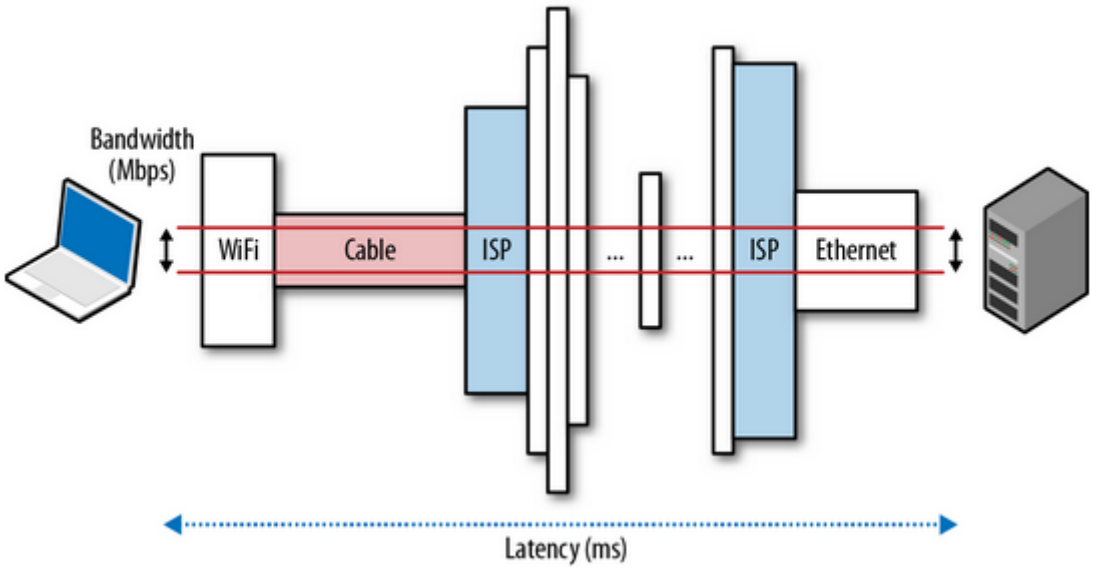
Verkon kaistanleveys on kasvanut nopeasti, mutta verkon viive ei ole pienentynyt samassa suhteessa. Verkon viiveeseen ei ole odotettavissa merkittäviä parannuksia, koska sitä hallitsee leviämisviive. Leviämisviive on aika, joka signaalilla kestää matkata lähettäjältä vastaanottajalle. Perustavanlaatuinen verkkoparametri Bandwidth-Delay Product (BDP) on huomattavasti suurempi nyt kuin Internetin alkuaikoina. BDP on datan määrä verkossa, joka on lähetetty, mutta ei vastaanotettu. Näyttää siltä, että HTTP/1.1 ja TCP eivät ole optimoituja verkoille, joissa BDP on suuri. (White et al. 2012)

Viiveeseen vaikuttaa oleellisesti myös etäisyys. Valonnopeus valokuitukaapelissa on noin kaksi kolmasosaa valonnopeudesta tyhjiössä, eli teoreettiselta maksimiltaan noin 200 000 km/s. Tällä nopeudella kestäisi 200 ms kiertää planeetta päiväntasaajalta. Verkon topologia vaikuttaa myös olennaisesti viiveeseen. Topologia antaa kaksi eri viivetyyppiä siihen, miten tieto kulkee verkon läpi. Ensimmäinen on viimeisen mailin pullonkaula. (Obren, Howell 2014) Iso osa viiveestä tulee viimeisillä kilometreillä. Internet-palveluntarjoaja joutuu reitittämään kaapeleita lähiöiden läpi saadakseen kodin yhdistettyä Internetiin. Muutamat ensimmäiset hypyt palveluntarjoajan pääreitittimille vievät usein jo kymmeniä millisekunteja. (Grigorik, 2013b) Toinen on viiveen pullonkaula, jossa kaukaisilla reitittimillä kestää siirtää toisilleen tietoa. Valokuidun keksiminen on jo huomattavasti pienentänyt viiveen pullonkaulaa ja merkittävästi kasvattanut lähiverkkojen nopeutta ja kapasiteettia. (Obren, Howell 2014)

Suoraan sanottuna matkalla on väliä. Vaikka data kulkee lähes valonnopeudella, niin pidempi matka lisää kulkuun käytettyä aikaa. Myös kun valo heikentyy kaapelissa, niin valoa täytyy tehostaa säännöllisesti ja se aiheuttaa viivettä. Esimerkiksi Uuden-Seelannin ja Pohjois-Amerikan välillä on noin 500 toistinta. Keskimääräinen kiertoaika (RTT) HTTP-pohjaiselle verkkosovellukselle kulkea Uudesta-Seelannista Englantiin on 375 ms. Itäiseen Aasiaan RTT on noin 350 ms ja läheiseen Australiaan 120 ms. Tätä voidaan verrata maailmanlaajuiseen keskiarvoiseen RTT:hen Googlen palvelimille, mikä on noin 100 ms. (Obren, Howell 2014)

Vuonna 2011 aloitettiin laskemaan Atlantin pohjalle uusi valokuituyhteys nimeltä Hibernia Express. Se yhdistää Lontoon ja New Yorkin toisiinsa ja sen ainoa tavoite oli säästää 5 ms viivettä verrattuna olemassa olleisiin linkkeihin luomalla suorempi reitti. Viive onkin hyvin merkittävä tekijä kaupantekoalgoritmeille rahoitusmarkkinoilla.(Grigorik, 2013b)

Verkkoliikenteen pääkomponentit ovatkin siis kaistanleveys ja viive. Kaistanleveys on maksimi suoritusteho kommunikaatiokaistalle. Viive on aika, joka kuluu paketin lähettämisestä vastaanottoon. Kuvassa 1.1 esitetään molemmat ominaisuudet. Kuvassa kaistanleveys on päätepisteiden välinen pienin kaistanleveys megabitteinä. Ei auta, jos matkalla on suuriakin kaistanleveyksiä, jos jokin vaihe kuristaa kaistanleveyden pieneksi. Viive on matkaan käytettävä aika millisekunteina. (Grigorik, 2013b)

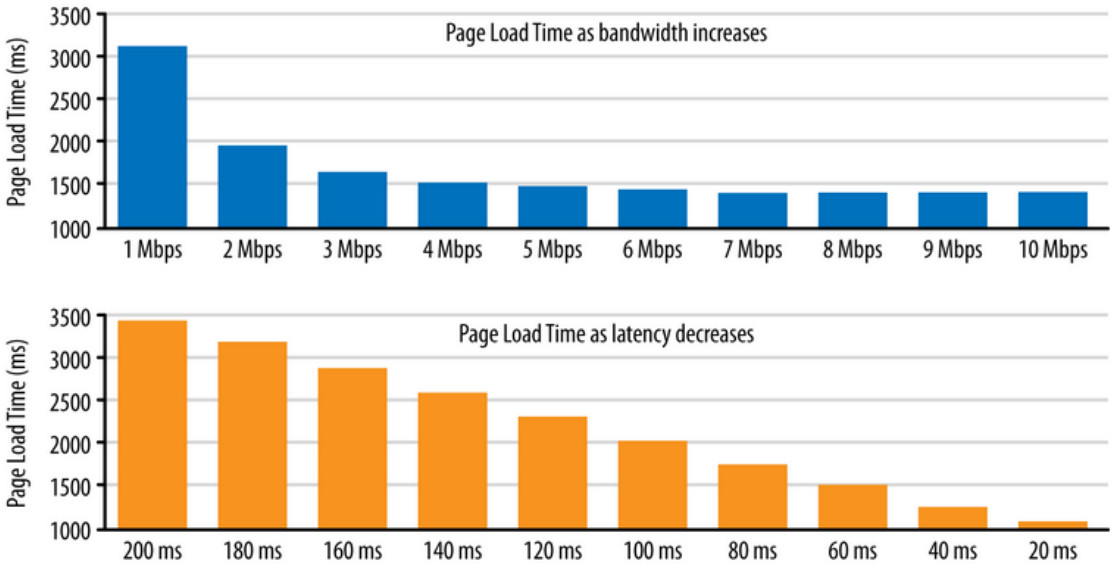


Kuva 1.1 Verkkoliikenteen kaistanleveys ja viive. (Grigorik, 2013b).

Ihmisten huomiokyky viiveelle on herkkä. Jo 100-200 ms viive vaikuttaa pätkivältä ja 300 ms vaikuttaa hitaalta. Tämä ei jätä verkolle tilaa virheille. Jotta kaikkea tietoa ei tarvitse hakea maapallon toiselta puolelta, ovat yritykseen jakaneet palvelimiaan joka puolelle. Näin strategisesti sijoittamalla palvelimet lähelle käyttäjiään voidaan saada huomattavia etuja. (Grigorik, 2013b)

Internet-palveluntarjoajat mainostavat aina suurempia kaistanleveyksiä ja muistuttavat sen monista hyödyistä, kuten nopeammista latauksista, lähetyksistä ja suoratoistosta. Suurempi määrä kaistanleveyttä on tietysti hyvä, varsinkin videon suoratoistossa ja muissa suurissa tiedonsiirroissa. Joka päiväisessä verkkoselauksessa haetaan kuitenkin satoja hyvin pieniä resursseja useasta eri paikasta, jolloin verkon kiertoajasta tulee rajoittava tekijä. (Grigorik, 2013b)

Minimoimaan viiveen vaikutusta käyttäjälle kehitettiin HTTP/2. Tämä saadaan aikaan otsake-kentän (header) kompressiolla ja käyttämällä useita samanaikaisia vaihtoja saman yhteyden sisällä. (Belshe et al. 2015) Alkusysäys HTTP/2- protokollan pohjana toimivan SPDY-protokollan kehittämiseen saatiin Mike Belshen tutkimuksesta (Grigorik, 2013b). Belshe toteaa kaistanleveyden tuplaamisen vaikuttavan alun jälkeen minimaalisesti verkkoselaukseen, kun taas verkon kiertoajan lasku nopeuttaa selausta aina (Belshe 2010). Kuvassa 1.2 havainnollistetaan kaistanleveyden kasvamisen ja viiveen pienentämisen seuraukset verkkoselaukseen.



Kuva 1.2 Kaistanleveyden kasvun ja viiveen vähentämisen vaikutus verkkosivun latausaikaan. (Grigorik, 2013b)

Tuleekin raja vastaan siinä, kuinka lähelle yleisöä palvelimet voidaan tuoda ja fysiikan lait siinä, kuinka nopeasti valo liikkuu kaapelissa. Tästä seuraa, että joudumme optimoimaan protokolliamme huomioimalla olemassa olevan kaistanleveyden- ja valon nopeuden rajoitukset. Tämä tapahtuu vähentämällä kiertomatkoja (round trips), siirtämällä data lähemmäs asiakkaita ja rakentamalla sovelluksia piilottamaan viive välimuistilla, ennakkohauilla ja vastaavilla tekniikoilla. (Grigorik, 2013b)

Verkon toiminnan tehostamiseksi vanhoille protokollille on tehty monia kiertoteitä verkkoselaimiin ja palvelimille. Joillain näillä kiertoteistä on vahingollisia seuraamuksia. Suurimmat seuraamukset tulevat useiden samanaikaisten TCP-yhteyksien avaamisesta.

Nykyaikaiset verkkoselaimen avaavat kuusi samanaikaista TCP-yhteyttä jokaiselle palvelimelle. Optimoidut verkkosivut hyväksikäyttävät tätä ominaisuutta jakamalla resurssejaan useammille palvelimille. Tämä ominaisuus tunnetaan domainin palasteluna (domain sharding). Palastelun seurauksena verkkoselaimella voi olla 20 tai jopa useampi TCP-yhteys avattuna yhden verkkosivun lataamiseen. Useiden yhteyksien avaamisella nopeutetaan verkkosivujen latausaikaa, mutta se voi hidastaa muita sovelluksia, jotka käyttävät vähän TCP-yhteyksiä. Tämä johtuu TCP:n tavasta jakaa kaistanleveyttä tasaisesti eri yhteyksille. (White et al. 2012)

Tämän diplomityön tarkoituksena on antaa yleiskuva HTTP/2-protokollan toiminnasta, vertailla HTTP-versioiden välistä tehokkuutta ja esitellä uuden version ominaisuuksia käytännössä. Tutkimus tehdään kirjallisuuskatsauksena. Kootaan jo olemassa olevia tutkimustuloksia yhteen ja pyritään selvittämään HTTP/2-protokollan nykyinen tilanne.

Luvussa 2 käydään läpi nykyisen verkon toimintaa esittelemällä TCP-protokolla ja HTTP-protokolla, HTTP/1.1:een liittyviä ongelmia ja HTTP/2:n pohjana olevan SPDY-protokollan toiminta. Luku 3 sisältää tarkempaa kuvausta HTTP/2:n sisältämistä parannuksista protokollaan ja sen ominaisuuksia. Luku 4 tarkastelee versioiden välistä nopeutta käytännön testeissä esittelemällä useita eri testitilanteita. Lopuksi luvussa 5 tehdään yhteenveto koko työhön.

2. HTTP/2 LÄHTÖKOHDAT

Nykyisen verkon yleisimmät protokollat ovat hypertekstin siirtoprotokolla (HTTP) ja kuljetuksenhallintaprotokolla TCP. TCP on luotettava tiedonsiirtoprotokolla, joka takaa toimituksen järjestyksessä, kopioiden suodatusta, vuonhallintaa ja muita kuljetusominaisuuksia kuljetuskerroksessa. HTTP on sovelluskerroksen protokolla, joka tarjoaa perussemantiikan pyynnöille ja vastauksille. (Google 2015)

Vuonna 2011 Pohjois-Amerikasta tehdyn tutkimuksen mukaan HTTP oli käytetyimpiä sovelluskerroksen protokollia. Varsinkin, kun reaaliaikainen viihde oli yli 50 % Internetliikenteestä ja monet niistä käyttävät HTTP:tä. Pelkkä Netflix oli yli 30 % koko liikenteestä. Muita huomattavia käyttöjä oli verkkoselaus (16.6 %) ja Youtube (9.9 %). (Mueller et al. 2015)

Modernit verkkoselaimet ovat keskittyneet kehittämään suorituskykyä. JavaScriptin toteutusnopeus jatkaa tasaista kasvua. Pelkästään vuonna 2012 mobiililaitteiden suorituskyky parani yli 50 %. Myös muut optimointitekniikat ovat helpottaneet piilottamaan verkon viivettä. Verkon viive onkin nyt usein pullonkaula verkkoselauksessa. (Grigorik 2013a)

Verkon rakenteen optimoinnilla tapahtuva kehitys on pysähtynyt viime vuosina. Kannettavien laitteiden yleistyessä on verkon viiveen pullonkaula vain pahentunut. 4G-laitteillakin puhutaan 50 millisekunnista ja vanhemmilla tekniikoilla sadoista millisekunneista. Jos tehokkuutta ei pystytä lisäämään parantamalla alla olevia linkkejä, niin huomio täytyy kääntää sovellusten rakentamiseen ja parantamaan siirtoprotokollien toimintaa. (Grigorik 2013a)

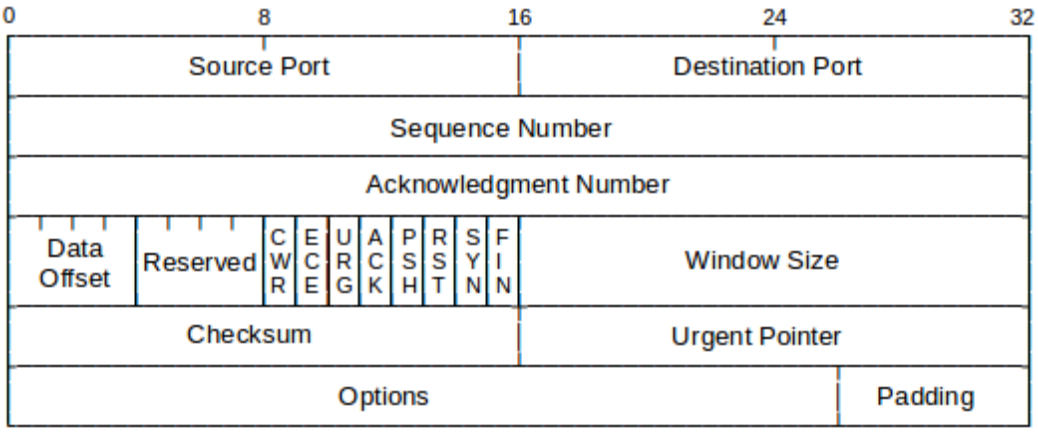
2.1 TCP-protokolla

TCP on suunniteltu tarjoamaan luotettavaa kommunikaatiota verkon yli prosessiparien kesken. TCP on yhteyspohjainen protokolla. TCP tarjoaa kaksi hyödyllistä palvelua datan etiketöintiin: työntö ja tärkeää. Normaalisti TCP päättää koska segmentissä on riittävästi dataa lähetettäväksi. Työnnössä (data stream push) TCP-käyttäjä voi vaatia TCP:tä lähettämään kaiken datan. Vastaanottopäässä TCP luovuttaa datan käyttäjälle samalla tavalla. Käyttäjä voi vaatia tätä, jos datassa on looginen tauko. Saapuva data voidaan myös merkitä tärkeäksi (urgent data signaling). Vastaanottaja päättää mitä se tällä tiedolla tekee. (Stallings 2000)

TCP-protokolla tarjoaa IP-protokollaa huomattavasti monipuolisempia palveluja. TCP käyttää vain yhtä protokollayksikköä, jonka nimi on TCP-segmentti. Tämän segmentin otsake löytyy kuvasta 2.1. Protokollan kaikki toiminnot täytyy pystyä tarjoamaan yhdessä

otsakkeessa, niin otsakkeesta tulee melko suuri ja pienimmilläänkin 20 oktettia. Otsakkeen osat esitellään listana: (Stallings 2000) (Dordal 2014)

- Lähettäjän TCP-portti (source port, 16 bittiä).
- Vastaanottajan TCP-portti (destination port, 16 bittiä).
- Sekvenssinumero (sequence number, 32 bittiä) ensimmäisestä dataoktetista segmentissä. Jos SYN-lippu on asetettu, niin ensimmäinen dataoktetti on alkusekvenssiluku +1.
- Kuittausluku (acknowledgement number, 32 bittiä) sisältää sekvenssinumeron seuraavasta dataoktetista, jonka TCP odottaa vastaanottavansa.
- Otsakkeen pituus (data offset, 4 bittiä) kertoo otsakkeessa olevien 32-bittisten sanojen määrän. Näin pystytään erottamaan otsake ja kehyksen sisältö.
- Varatut kentät (reserved, 6 bittiä). Mahdollista tulevaa tarvetta varten.
- Seuraavaksi on kahdeksan pientä lippukenttää. CWR ja ECE ovat ruuhkan ilmoitusmekanismi, URG on tärkeän datan ilmoitus, ACK kertoo kuittausluvun olevan tärkeä ja yhteydenmuodostuksen jälkeen tämä lippu pitäisi olla aina asetettu, PSH kertoo työnnettävien pakettien määrän, RST ilmoittaa virhetiloista, SYN merkitsee yhteydenmuodostamisessa käytettävät paketit ja FIN merkitsee yhteyden lopettamiseen käytettävät paketit.
- Vuon ohjauksen krediittien sijoitus oktetteina (window, 16 bittiä). Dataoktettien määrä, jonka lähettäjä on valmis vastaanottamaan alkaen kuittauslukukentän ilmoittamasta oktetista.
- Tarkistussumma (checksum, 16 bittiä).
- Kiireellisyysosoitin (urgent pointer, 16 bittiä) kertoo kiireellisen datan viimeisen oktetin, jotta vastaanottaja tietää kiireellisen datan määrän.
- Valinnainen (options) sisältää esimerkiksi maksimikoon vastaanotettavalle segmentille
- Täyte (padding) täyttää otsakkeen lopun, niin että otsake loppuu 32 bitin kohdalle.



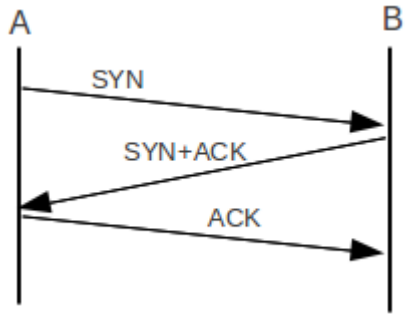
Kuva 2.1 TCP-otsake. (Dordal 2014)

TCP-protokolla on suunniteltu toimimaan IP-protokollan kanssa. Joitakin TCP:n parametreja annetaan alaspäin IP:lle käytettäväksi sen otsakkeessa. Tästä TCP:n ja IP:n linkistä seuraa, että minimimäärä protokollien tarvitsemaan ohjausliikenteeseen on itseasiassa 40 oktettia. (Stallings 2000)

Seuraavaksi käydään läpi TCP:n ominaisuuksia. TCP on vuopainotteinen, eli sovellus voi kirjoittaa dataa vähän tai hyvin paljon ja antaa sen TCP:lle paketoitavaksi. TCP on yhteyspainotteinen, eli yhteys täytyy luoda ennen kuin mitään dataa voidaan siirtää. TCP on luotettava, sillä se pitää sekvenssinumeroineen huolta, että data saapuu oikeassa järjestyksessä. Uudelleenlähetysmekanismit pitävät huolta, että dataa ei menetetä. Maksimaalisen tiedonsiirron saavuttamiseksi TCP käyttää liikkuvaa ikkuna-algoritmia. Nämä ominaisuudet tekevät TCP:stä hyvin sopivan suurien tiedostojen siirtoon. Päätepisteet avaavat yhteyden ja data siirtyy luotettavasti niiden välillä. (Dordal 2014)

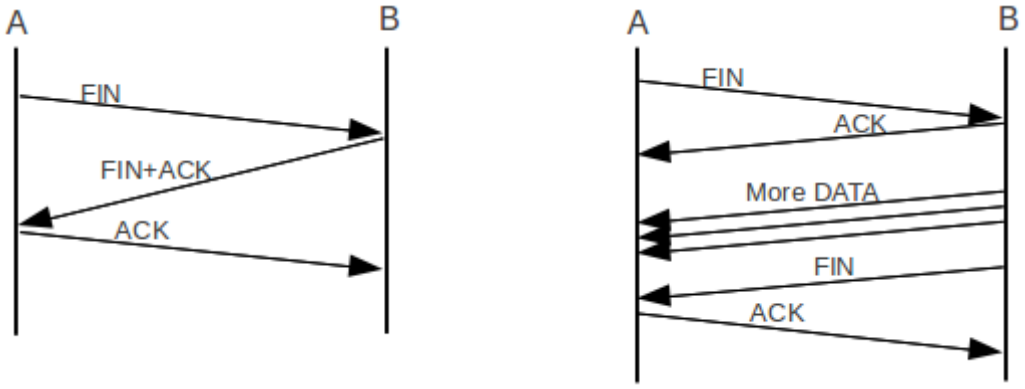
TCP:n oma vuonhallintamekanismi suunniteltiin mahdollistamaan, että vastaanottaja voi hallita vastaanotettavan datan määrää, jotta sen vastaanottopuskuri ei täyty. Nykyään tätä samaa vuonhallintamekanismia käytetään erilaisin tavoin tarjoamaan ruuhkanhallintaa Internetin yli. Ruuhkautumisella on kaksi suurempaa vaikutusta. Ensinnäkin ruuhkautumisen alussa lähetysaika verkon yli kasvaa. Ruuhkan kasvaessa verkon välittäjät alkavat tiputtaa paketteja ja segmenttejä pois. TCP:n vuohallintaa voidaan käyttää tunnistamaan ruuhkautuminen ja reagoimaan siihen vähentämällä lähetettävän datan määrää. Jos monet Internetissä toimivat entiteetit noudattavat hillintää, niin Internetin ruuhka helpottuu. (Stallings 2000)

TCP:n yhteydenmuodostus tunnetaan kolmivaihekättelynä. Yhteys muodostetaan kuvan 2.2 mukaisesti. A on asiakas ja B on kuunteleva palvelin. A lähettää SYN-paketin B:lle. SYN-paketissa on SYN-lippu asetettuna. B vastaa tähän omalla SYN-paketilla, jossa on myös ACK lippu asetettuna. A vastaa nyt pelkällä ACK-paketilla. Nyt yhteys on muodostunut. (Dordal 2014) Huomattavaa on, että yhteys muodostetaan lähetys- ja vastaanottoporttien välille. Näin ollen vain yksi yhteys voi olla auki samojen porttien välillä. Jos portti tukee useita yhteyksiä, niin se voi avata haluamansa määrän yhteyksiä, kunhan ne ovat eri vastaanottoportteihin. (Stallings 2000)



Kuva 2.2 TCP-yhteyden muodostamisen kolmivaiheinen kättely. (Dordal 2014)

Yhteyden sulkemiseksi tehdään avaamista vastaava toimenpide FIN-paketeilla. Lopetus on oikeastaan kaksi erilaista kättelyä sen mukaisesti, onko toinen osapuoli valmis sulkemaan yhteyden heti. Kuvassa 2.3 esitetään kaksi eri tilannetta. Vasemmalla A lähettää B:lle paketin, jossa FIN-lippu on asetettu ja näin ilmoittaa datan lähetyksen päättyneen omalta osaltaan. B lähettää A:lle FIN-paketin, jossa on ACK-lippu asetettu. A vastaa ACK-paketilla ja yhteys lopetetaan. Oikealla B ei ole valmis päättämään yhteyttä heti saatuaan lopetuspyynnön A:lta. Nyt B vastaa vain ACK-paketilla ja lähettää haluamansa määrän dataa A:lle ja ollessaan valmis lopettamaan lähetyksen, niin B lähettää A:lle oman FIN-pakettinsa. A vastaa ACK-paketilla ja yhteys lopetetaan. FIN-paketti onkin oikeastaan lupaus olla lähettämättä enää dataa. (Dordal 2014)



Kuva 2.3 TCP-yhteyden sulkeminen. (Dordal 2014)

2.2 HTTP:n historia

HTTP on yksi Internetin käytetyimmistä sovelluskerroksen protokollista. Se on yleinen kieli asiakkaan ja palvelimen välillä mahdollistaen modernin verkon. Yksirivisestä käskystä se on kehittynyt protokollaksi, jota käytetään verkkoselaimien lisäksi useissa Internetissä kiinni olevassa sovelluksessa. (Grigorik 2013b)

HTTP/0.9 oli ensimmäinen versio HTTP:stä. Vuonna 1991 Tim Berners-Lee hahmotteli uuden protokollan pääpiirteet ja listasi useita suunnittelutavoitteita. Näitä tavoitteita olivat: tiedoston siirto, kyky pyytää luetteloitua tietoa hypertekstihakemistosta, formaatin neuvottelu ja kyky ohjata asiakas eteenpäin toiselle palvelimelle. Teorian todistamiseksi rakennettiin prototyyppi, joka toteutti pienen osan ehdotetusta toiminnallisuudesta: (Grigorik 2013b)

- Asiakas pyytää yhden ASCII-merkkisarjan.
- Asiakkaan pyyntö lopetetaan rivivaihdolla.
- Palvelimen vastaus on ASCII-merkkivuo.
- Palvelimen vastaus on HTML-muodossa.
- Yhteys lopetetaan, kun dokumentin siirto on valmis.

Käytännössä tämä tarkoitti yksirivistä GET-pyyntöä ja osoitinta pyydetyn dokumentin sijainnille. Vastauksena saatiin yksi hypertekstidokumentti ilman otsakkeita tai muuta tietoa. HTTP-protokolla oli syntynyt. (Grigorik 2013b)

HTTP/1.0 kehittyi vuosien 1991 ja 1995 välillä yhdessä verkkoselaimien kanssa. Lista toivottuja ominaisuuksia HTTP:lle kasvoi ja paljasti HTTP/0.9:n rajoittuneisuuden. Tarvittiin protokolla siirtämään pelkän hypertekstidokumentin lisäksi myös muuta dataa ja kykyä tarjota metadataa pyynnöistä ja vastauksista. HTTP:n kehittyminen oli tässä vaiheessa ad hoc -prosessi. Uusia ominaisuuksia lisättiin ja julkistettiin, jonka jälkeen katsottiin, jos muutkin alkaisivat käyttää niitä ominaisuuksia. (Grigorik 2013b)

Tämän nopean kokeiluajanjakson aikana parhaat toimintatavat ja säännönmukaiset kuviot alkoivat paljastua. Vuonna 1996 HTTP Working Group julkisti RFC 1945:n, joka dokumentoi normaaleja käyttötapoja eri HTTP/1.0-toteutuksille. Huomattavaa on, että tämä ei ole virallinen Internet-standardi, vaan tiedoksianto. (Grigorik 2013b)

Huomattavia muutoksia HTTP-protokollaan olivat: (Grigorik 2013b)

- Pyyntö voi sisältää useita erillisiä otsakekenttiä.
- Vastauskohteella on oma määrä uusien rivien erottamia otsakekenttiä.
- Vastauskohteen ei tarvitse olla enää vain hypertekstiä.
- Palvelimen ja asiakkaan välinen yhteys suljetaan jokaisen pyynnön jälkeen.

Pyynnöt ja vastaukset pidettiin ASCII-koodattuna, mutta vastauksen kohde voi olla mitä tahansa tyyppiä. Kohde voi olla HTML-tiedosto, pelkkää tekstiä, kuva tai mikä tahansa sisältö. Näin jo protokollan nimi olisi voinut muuttua, koska enää ei siirretä pelkkää hypertekstiä, mutta alkuperäinen nimi pysyi. (Grigorik 2013b)

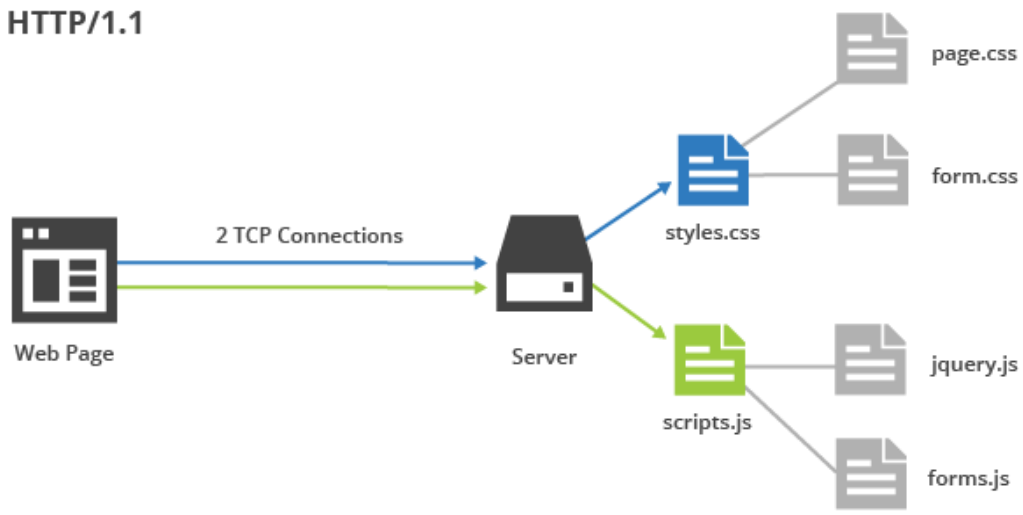
Työ HTTP:n standarditoimiseksi versioksi HTTP/1.1 tapahtui noin neljän vuoden aikana vuodesta 1995 vuoteen 1999. Ensimmäinen standardi RFC 2068 julkistettiin vuonna 1997. Kaksi ja puoli vuotta myöhemmin, vuonna 1999, parannusten ja päivitysten kanssa julkistettiin uusi RFC 2616. (Grigorik 2013b) Työ HTTP/1.1:n kanssa ei kuitenkaan loppunut, vaan uusi versio RFC 7230 julkistettiin vielä kesäkuussa 2014 (Fielding, Reschke 2014).

HTTP/1.1-standardi ratkaisi monta protokollaa vaivannutta monitulkintaisuutta ja esitteli useita tehokkuusoptimointeja. Tehokkuutta lisäämään tulivat hengissä pysyvät yhteydet (keepalive connections), lohkotut koodaussiirrot (chunked encoding transfers), tavualuepyynnöt (byte-range requests), pyyntöjen ketjuttamiset (pipelining) ja paljon muuta. (Grigorik 2013b)

Alla esitellään nimetyt parannukset:

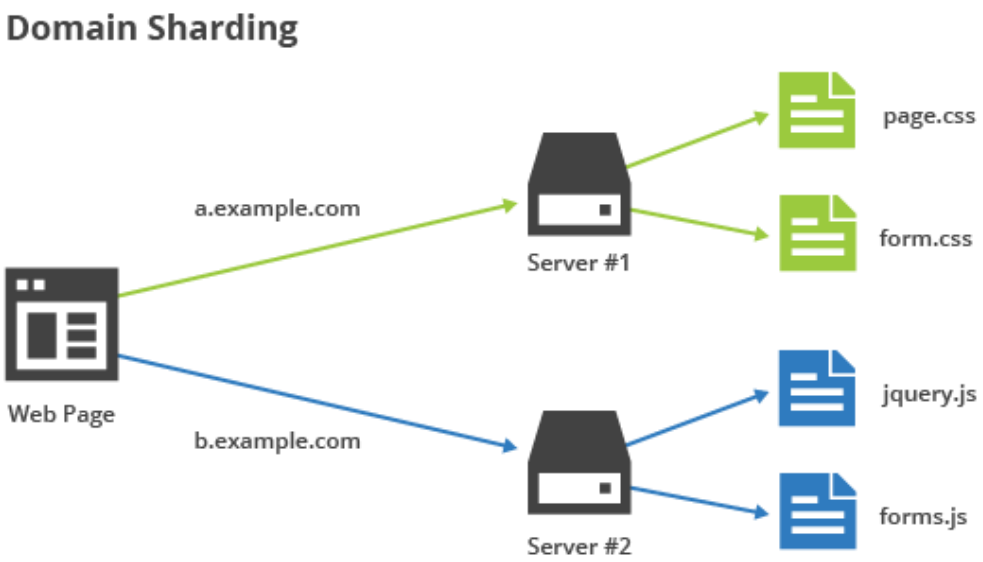
- Hengissä pysyvän yhteyden avulla samaa TCP-yhteyttä pystytään uudelleenkäyttämään. HTTP/1.1:ssä hengissä pysyvä yhteys asetettiin oletustoiminnaksi. Näin palvelimen pitäisi pitää yhteyttä auki, kunnes se suljetaan erikseen. (Grigorik 2013b)
- Lohkottu siirto poistaa viestin rungon rajoituksen. Tämä on mahdollista jakamalla viesti lohkoihin. Jokaista lohkoa ennen lähetetään lohkon koko vastaanottajalle. Lohko loppuu, kun vastaanottaja saa nollan mittaisen lohkon ilmoituksen. (Krishnamurthy et al. 1999)
- Tavualuepyynnössä asiakas voi pyytää vain osaa resurssista. HTTP/1.1:ssä tämä osa määritetään bitteinä. Asiakas voi haluta vain dokumentin alun tai jatkaa kesken jäänyttä latausta. (Krishnamurthy et al. 1999)
- Pyyntöjen ketjuttamisessa asiakas ei odota palvelimelta vastausta ennen kuin se lähettää uuden pyynnön samassa yhteydessä. Pyynnöt on edelleen pakko lähettää ja vastaanottaa järjestyksessä, mutta tämä toiminta käyttää TCP-yhteyttä parhaalla mahdollisella tavalla. (Krishnamurthy et al. 1999)

Monia tapoja kehitettiin nopeuttamaan HTTP/1.1 toimintaa myös palvelimien päässä. Käytettyjä nopeuttamiskeinoja ovat esimerkiksi liitostaminen, palastelu ja spritejen yhdistäminen. Nämä keinot esitellään seuraavaksi. Usein liitostettiin (concatenate) kaikki tyylitiedostot (CSS) samaan tiedostoon. Samoin voitiin tehdä JS-tiedostoille. Tämä johtaa HTTP-pyyntöjen määrän selvään laskuun ja näin merkittävästi parantaa HTTP/1.1:n toimintaa. Kuvassa 2.4 näkyy liitostaminen selvästi. Palvelin liitostaa kaikki CSS-tiedostonsa yhteen ja näin tarvitaan vain yksi TCP-yhteys hakemaan ne. Myös JS-tiedostot on liitostettu toisiinsa ja vaatii vain yhden TCP-yhteyden. (Hodson 2015)



Kuva 2.4 HTTP/1.1-tiedostojen liitostaminen yhteen. (Hodson 2015)

Yleisesti käytetään myös domainin palastelua. Palastelun tarkoituksena on huijata selainta avaamaan enemmän TCP-yhteyksiä kuin sen pitäisi. Palastelu nopeuttaa HTTP/1.1:n toimintaa, mutta sillä on myös haittavaikutuksia. Jokainen pala vaatii oman DNS-haun, TCP-yhteyden ja TLS-kättelyn, jos palvelimet käyttävät eri TLS-sertifikaattia. Kuvassa 2.5 löytyy verkkosivu, jonka tiedot on jaettu kahdelle eri palvelimelle. Näin selain avaa omat TCP-yhteydet sekä palvelimelle yksi että palvelimelle kaksi. (Hodson 2015)

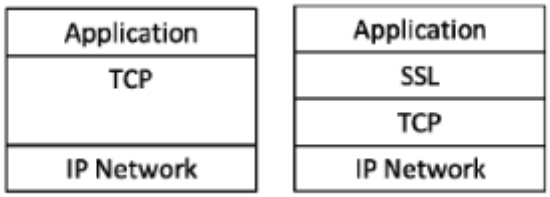


Kuva 2.5 Domainin palastelu TCP-yhteyksien määrän lisäämiseksi. (Hodson 2015)

Spritejen yhdistämiseksi kutsutaan tapaa yhdistää pieniä kuvia yhdeksi suureksi kuvaksi. Tämän jälkeen JS:n tai CSS:n avulla leikataan isosta kuvasta tarvittava osa näyttämään vain haluttu kuva. Tämä nopeuttaa lataamista HTTP/1.1:n kanssa, koska yksi kuva on huomattavasti nopeampi hakea kuin sata pientä kuvaa. Kuvasta 2.6 löytyy esimerkki spritejen yhdistämisestä. (Stenberg 2015)

Kuva 2.6 Pienten kuvien yhdistäminen yhdeksi suureksi kuvaksi. (Stenberg 2015)

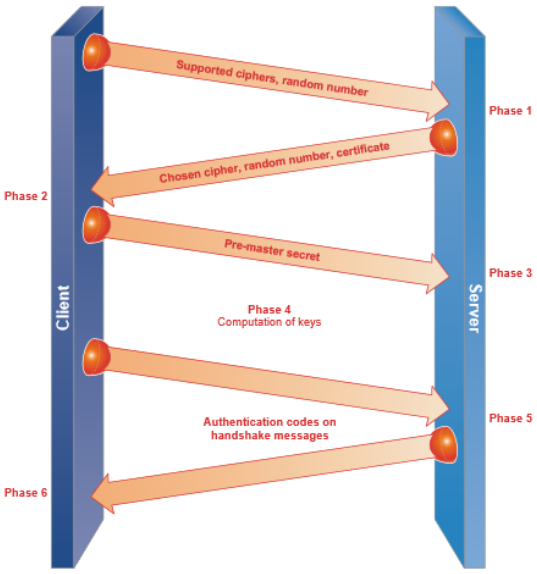
Internetissä on entistä enemmän kaupallista ja tärkeää toimintaa, joten tietoturva on tullut tarpeellisemmaksi. Näin HTTP:sta on oma HTTPS-versio, johon tietoturva on lisätty kuljetuskerroksen ja sovelluskerroksen välille. SSL-tietoturvakerros tarjoaa luotettavuutta, koskemattomuutta ja päätepisteiden välisen todentamisen. Kuvasta 2.7 näkyy SSL-kerroksen lisääminen. (Allison, Bakri 2015)



Kuva 2.7 SSL-suojauskerroksen lisäys kuljetus- ja sovelluskerroksen väliin. (Allison, Bakri 2015)

Monet yleiset verkkosovellukset ovat siirtyneet käyttämään niiden suojattua SSL-versiota. Kuitenkin hyvin pieni osa HTTP-liikenteestä on salattua. Toukokuussa 2014 olleen Snowden-tapauksen johdosta tuli kuitenkin selvä piikki salauksen käyttöön. Tällä viitataan siis Edward Snowdenin paljastamaan laajamittaiseen Internetin vakoiluun, jota Yhdysvaltojen tiedusteluelin toteutti. Ruuhka-aikana Euroopassa suojatun liikenteen määrä nousi 1.47 prosentista 6.10 prosenttiin. Vastaavasti Amerikassa noustiin 2.29 prosentista 3.80 prosenttiin ja Latinalaisessa Amerikassa 1.80 prosentista jopa 10.37 prosenttiin. Vaikka HTTPS tarjoaa hyvää turvallisuutta, niin ongelmat sen julkisen avaimen perusrakenteessa (PKI) ovat osittain laskeneet sen luotettavuutta. (Allison, Bakri 2015) (Sandvine 2014)

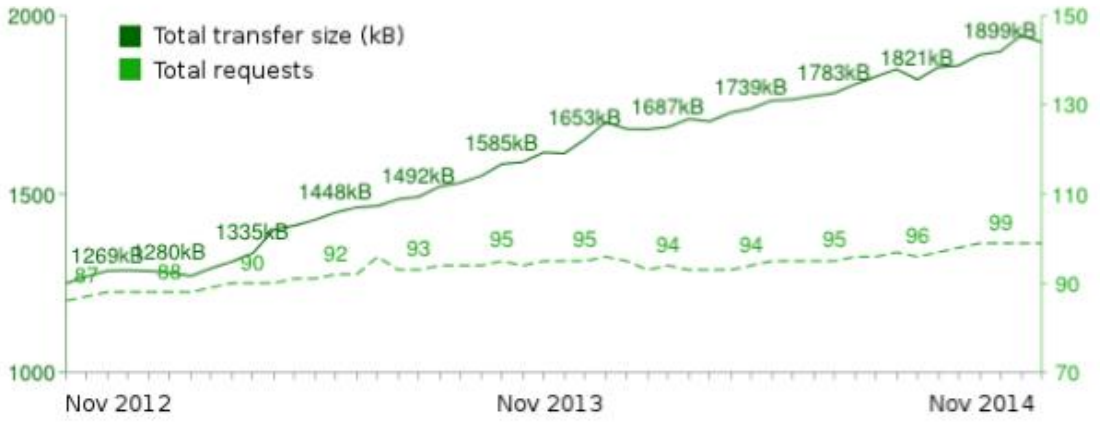
SSL:n käyttö pidentää huomattavasti HTTP-kättelyä. Kättely sisältää kättelyvaiheen ja salausdatan vaihdon. Kättely löytyy kuvasta 2.8. Kättely alkaa, kun asiakas lähettää palvelimelle listan tuetuista algoritmeista ja satunnaisluvun avaimen luomista varten. Vaiheessa kaksi palvelin valitsee salauksen ja lähettää sen takaisin oman palvelimensa yleisen avaimen toisen satunnaisluvun kanssa. Vaiheessa kolme asiakas varmistaa palvelimen sertifikaatin ja purkaa palvelimen julkisen avaimen. Sen jälkeen asiakas luo satunnaisen merkkijonon salausta varten ja salaa sen palvelimen julkisella avaimella. Tämä salattu merkkijono lähetetään palvelimelle. Vaihe neljä vie SSL-prosessissa eniten laskentatehoa palvelimelta, kun se purkaa salatun merkkijonon. Asiakas ja palvelin luovat molemmat itsenäisesti merkkijonosta istunnossa käytettävän salausavaimen. Vaiheessa viisi ja kuusi käydään läpi normaali kättely salattuna. (Grover et al. 2009)



Kuva 2.8 SSL-salauksen kättely. (Grover et al. 2009)

2.3 HTTP:n ongelmia

HTTP:ta ei ole suunniteltu toimimaan viiveen kanssa. Verkkosivut ovat nykyään hyvin erilaisia kuin ne ennen olivat tai millaisiksi niiden odotettiin kehittyvän. (Google 2015) Useimmat verkkosivut sisälsivät vähän tekstiä ja pieniä kuvia. Verkkosivuilla on nykyään paljon enemmän dataa ja datan määrä kasvaa koko ajan, joten verkkosivut ovat tulleet paljon resurssi-intensiivisemmiksi. (Kim et al. 2015) Verkkosivut myös tunnistavat jokaisen painalluksen ja hiiren liikkeen verkkosivun mallinnukseen. Tämän kaiken hintana on raskas verkkoliikenne asiakkaan ja palvelimen välillä. Kuvasta 2.9 näkyy, että pyyntöjen määrä ja keskimääräinen koko ovat kasvaneet rajusti. Kehityksen odotetaan jatkuvan seuraavina vuosina. (de Saxcé et al. 2015)



Kuva 2.9 Pyyntöjen määrän ja keskiarvoisen koon kehitys. (de Saxcé et al. 2015)

Suuri osa HTTP-liikenteestä koostuu pienistä purskeista dataa, jota siirretään kymmenien TCP-yhteyksien yli. Kuitenkin TCP on optimoitu kauan kestäviin yhteyksiin ja siirtämään paljon dataa. Verkon kiertoaika on rajoittava tekijä uusille TCP-yhteyksille sen ruuhkanhallinnan takia. Verkon kiertoaika on asiakkaan pyynnön lähetyksestä ensimmäiseen vastaukseen kuluva aika. Myös viive on pullonkaula useimmille verkkosovelluksille. (Grigorik 2013a)

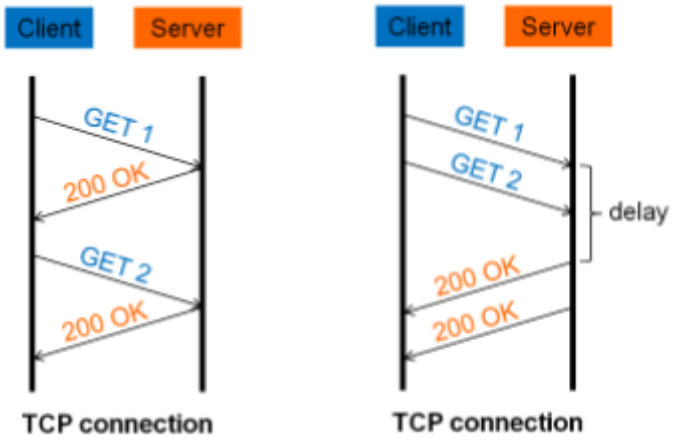
HTTP:llä on useita ongelmia: (Google 2015)

- Yksi pyyntö yhdelle yhteydelle. 500 ms viive palvelimella estää TCP-yhteyden uudelleenkäytön.
- Vain asiakas pystyy aloittamaan pyynnön. Vaikka palvelin tietää mitä asiakas tarvitsee seuraavaksi, niin se joutuu odottamaan asiakkaan pyytävän sitä erikseen.
- Pakkaamattomat pyyntö- ja vastausotsikot. Esimerkiksi ADSL-modeemeilla on melko pieni lähetysnopeus, joten viive voi olla huomattava.
- Toistuvat otsikot. Monet otsikot ovat muuttumattomia ja niitä ei tarvitsisi lähettää uudelleen.
- Valinnainen datan pakkaus. Sisältö pitäisi aina lähettää pakattuna.

Perustoiminnaltaan HTTP:ssa asiakas lähettää GET-pyyntöjä palvelimelle saadakseen sisältöä. Jos asiakas haluaa useita sisältöjä, niin joudutaan lähettämään yhtä monta GET-pyyntöä. Kuitenkin ennen kuin uutta pyyntöä pystytään lähettämään, täytyy odottaa palvelimelta vastaus edelliseen pyyntöön. Näin ollen sivun lataamisaika on lähellä pyyntöjen määrää kerrottuna asiakkaan ja palvelimen välisellä viiveellä. Tähän ongelmaan kehitettiin jo HTTP/1.1 ratkaisu nimeltä ketjutus (pipelining). (de Saxcé et al. 2015)

Kuvassa 2.10 esitellään HTTP:n perustoiminnon ja ketjutuksen ero. Oikealla puolella kuvassa asiakas lähettää kaksi pyyntöä (GET 1 ja GET 2) odottamatta palvelimelta vastausta

ja saa vastaukset samassa järjestyksessä. Tämä on jo huomattava parannus perustoimintoon (kuvassa vasemmalla), jossa jokaiseen GET-pyyntöön tarvitaan vastaus. (de Saxcé et al. 2015)



Kuva 2.10 HTTP:n perustoiminto vasemmalla ja ketjutus oikealla. (de Saxcé et al. 2015)

Ketjutus esittelee kuitenkin uuden ongelman: head-of-line blocking (HOL). Kyseessä on HTTP/1.1:n ketjutusongelma, jossa yksi odottava vastaus voi viivyttää useita muita vastauksia. Ongelma on oikeassa järjestyksessä lähettäminen, sillä väärässä järjestyksessä tulleet paketit odottavat kunnes ensimmäinen vastaus saapuu. (Mueller et al. 2015) Esimerkkinä suurikokoiset vastaukset, joiden lähetys kestää pidemmän aikaa, viivästyttävät sen jälkeen tulevia vastauksia. (de Saxcé et al. 2015)

HTTP:n nopeuttamiselle on tehty monia ehdotuksia. Monet niistä ovat keskittyneet kuljetus- ja istuntokerroksille. Kuljetuskerroksen protokollan vaihto olisi hyvin vaikea toteuttaa. On huomattavasti helpompaa korjata HTTP:n omat ongelmat, mikä vaatii hyvin pieniä muutoksia olemassa olevaan infrastruktuuriin. (Google 2015)

2.4 SPDY-protokolla

HTTP/2:n pohjaksi tarjottiin kolmea ehdotusta: Googlen SPDY, Microsoftin Speed+Mobility ja verkkoystävällinen HTTP päivitys. Näistä SPDY valittiin uuden protokollan pohjaksi. (Mueller et al. 2015) Speed+Mobility:n tarkoituksena on korostaa toiminnan parannuksia ja turvallisuutta. Samaan aikaan otetaan huomioon myös mobiililaitteiden tarpeet. (Trace et al. 2012) Verkkoystävällinen HTTP päivitys pyrkii olemaan nopeampi, luotettavampi ja ystävällisempi mobiiliverkoille. Myös semantiikan säilytys ja yhteensopivuuden säilyttäminen halutulla tasolla olivat tavoitteina. (Tarreau et al. 2012)

Vuonna 2009 Google alkoi kehittää HTTP:n semantiikan säilyttävää protokollaa, joka korjaa HTTP/1.1:tä haitanneen HOL-ongelman. Sen nimi on SPDY (lausutaan speedy).

(de Saxcé et al. 2015) SPDY toimii kuljetuskerrosprotokolla TCP:n päällä. SPDY ylläpitää vain yhtä yhteyttä. Tämä on suuri ero HTTP:hen, joka ei sallinut jatkuvia yhteyksiä. Yhdellä yhteydellä käytetään vuota (stream), johon kaikki viestit kanavoidaan (multiplexing). Istunnon aikana voidaan avata useita kahdensuuntaisia (full-duplex) yhteyksiä vuon sisään. Palvelin ylläpitää yhteyttä aikakatkaisuun asti tai kunnes asiakas katkaisee sen. (Mueller et al. 2015)

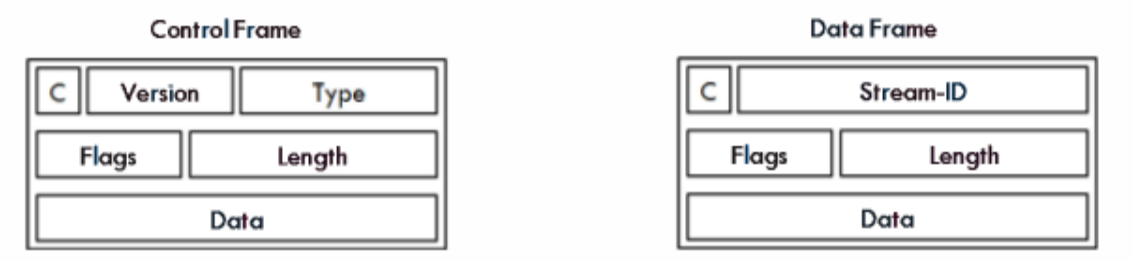
SPDY:n kanssa asiakas avaa vain yhden TCP-yhteyden jokaiselle palvelimelle ja pyytää palvelimen välittämään kaiken tiedon tämän yhteyden sisällä. Vain yhden TCP-yhteyden käyttö auttaa TCP:n oman ruuhkanhallinta-algoritmin toimintaa. Näin TCP pystyy tasaisesti hallitsemaan dataliikennettä verkossa. Asiakas pystyy sisällyttämään useita pyyntöjä yhteen viestiin ja näin vähentämään ylimääräisen otsaketiedon lähettämistä, sekä ylimääräisten kiertoaikojen määrää ennen lähetyksen aloittamista. (White et al. 2012)

Pyyntöjen priorisointi on yksi SPDY:n ominaisuuksista. Viestien kanavoinnista johtuu, että verkkosivun kriittisiä resursseja ladataan hitaammin, kun vähemmän tärkeitä ladataan samanaikaisesti. Tämän estämiseksi SPDY mahdollistaa asiakkaan ilmoittamaan suhteellinen prioriteetti pyydetylle resurssille. Näin esimerkiksi JavaScript-kirjasto voidaan ladata ensin luomaan URL:t sivun muille resursseille. (White et al. 2012)

SPDY mahdollistaa HTTP-otsakkeen pakkauksen vähentämään lähetettävää dataa. Otsakkeen pakkaus on ollut standardi jo aikaisemmin, mutta vasta SPDY tekee sen käytöstä pakollista. Vastauksille tästä ei ole niin paljon hyötyä kuin pyynnöille. Jokaisessa pyynnössä samalta asiakkaalta samalle palvelimelle voi olla sama tunniste. Tunnisteen koko voi olla yli 100 bittiä. Chromiumin ja Firefoxin kehittäjät ovat raportoineet noin 90 % pakkausta. Tämä voi mahdollistaa useampia pyyntöjä pakattavaksi yhteen pyyntöpakettiin. (White et al. 2012)

SPDY on HTTP-yhteensopiva ja voitaisiin integroida istuntokerrokseen HTTP:n ja TCP:n väliin. SPDY tarjoaa rajapinnan HTTP:lle, mikä yksinkertaistaa integrointia olemassa oleville HTTP-sovelluksille. (Mueller et al. 2015)

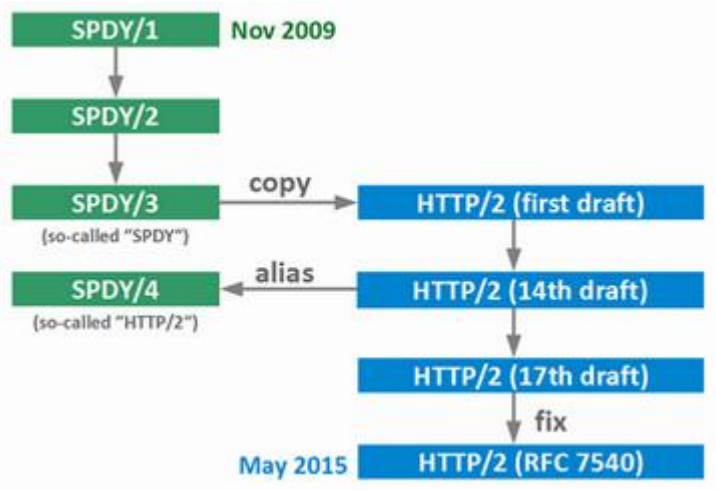
SPDY perustuu kehyksiin, joita asiakas ja palvelin vaihtavat. SPDY:ssä on kaksi kehystyyppiä. Ne ovat kontrollikehys (control frame) ja datakehys (data frame). Kehykset esitellään kuvassa 2.11. Kehyksen tyypin pystyy tunnistamaan alussa olevasta kontrollibitistä (C). Kontrollikehyksessä on SPDY:n version tunnistekenttä (version), joka kertoo käytettävän SPDY:n version. Tyyppikenttä (type) kertoo millainen kontrollikenttä on kyseessä. SPDY käyttää kahdeksaa erilaista kontrollityyppiä. Versio- ja tyyppikentän sijaan datakehyksessä on vuon tunnistekenttä (stream-ID), joka vain kertoo, mihin vuohon datakehys kuuluu. Tätä tarvitaan tunnistamaan vuo, sillä useita voita voi olla auki samanaikaisesti. Lippukenttä on kontrollikehyksessä tyyppiriippuvainen, kun taas datakehyksessä se voi olla vain vuon lopetusilmoitus. Näin ei tarvita uutta kierrosta vuon lopettamiseksi. (Mueller et al. 2015)



Kuva 2.11 SPDY-kehystyypit. (Mueller et al. 2015)

SPDY:n vuot ovat palvelimen tai asiakkaan luomia kehysten sarjoja. Vuot ovat kaksisuuntaisia. HTTP toimii SPDY:n päällä normaalisti ja luo oman vuon jokaiselle pyynnölleen. Tämä ei kuitenkaan vaikuta tehokkuuteen millään tavalla, sillä uuden vuon voi luoda ilman ylimääräistä kiertoaikaa. (Mueller et al. 2015)

Internet Engineering Task Force -standardointiorganisaatio sai toukokuussa 2015 valmiiksi RFC 7540 -standardin HTTP/2:sta (Torii 2016). Kuvasta 2.12 löytyy SPDY:n eri kehitysversioiden muuttuminen lopulta HTTP/2:ksi.



Kuva 2.12 SPDY:n kehityskaari HTTP/2:ksi. (Torii 2016)

3. HTTP/2-PARANNUKSET JA RAKENNE

Tässä luvussa käydään läpi uusitun hypertekstin siirtoprotokollan rakenne ja tehdyt parannukset. Se tunnetaan hypertekstin siirtoprotokollan versiona 2. HTTP/2 optimoi verkon resurssien käyttöä minimoimalla verkon viiveen vaikutusta käyttäjälle. Tämä saadaan aikaan otsake-kentän (header) kompressiolla ja käyttämällä useita samanaikaisia vaihtoja saman yhteyden sisällä. Samalla esitellään palvelimelle mahdollisuus työntää (push) dataa asiakkaalle. (Belshe et al. 2015)

HTTP/2 nimettiin kokonaan uutena versiona 2 sen sijaan, että sitä olisi kutsuttu nimellä HTTP/1.2. Tämä johtuu siitä, että se ei ole taaksepäin yhteensopiva aikaisempien HTTP/1.x versioiden kanssa. (Grigorik, 2013b)

HTTP/2 kehitettiin, koska aikaisemmassa versiossa 1.1 on monia ominaisuuksia, joilla on negatiivinen vaikutus sovellusten tehokkuuteen. HTTP/1.0 ja HTTP/1.1 ovat asiakkaita, joiden tarvitsee tehdä useampia pyyntöjä palvelimelle saavuttaakseen samanaikaisuuden ja vähentämään viivettä. Myös HTTP:n otsake-kentät ovat usein toistuvia, mikä aiheuttaa tarpeetonta verkkoliikennettä ja aiheuttaa TCP-ruuhkaikkunan täyttymisen. Tämä aiheuttaa viivettä, kun useita pyyntöjä tehdään uudella TCP-yhteydellä. (Belshe et al. 2015)

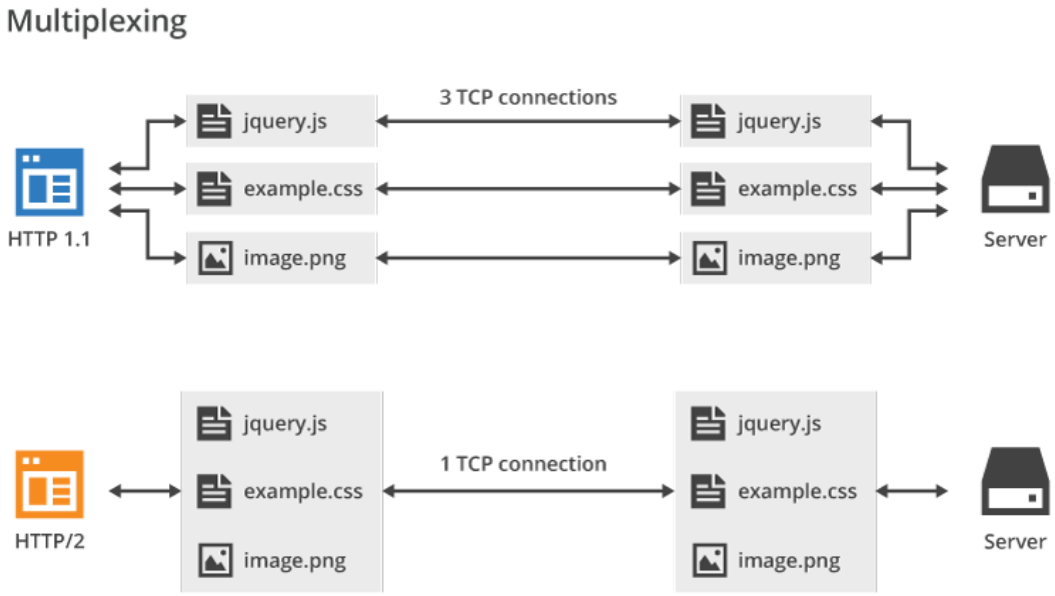
HTTP/2 pyrkii korjaamaan näitä vikoja optimisoimalla HTTP:n semantiikkaa alla olevaan yhteyteen. Tämä toteutetaan sallimalla saman yhteyden yli pyyntö- ja vastausviestit ja koodaamalla otsakekentät tehokkaasti. (Belshe et al. 2015) HTTP/2-standardin kanssa käytetään HPACK-nimistä algoritmia, joka määrittää, kuinka otsakkeet pakataan. HPACK-mekanismi käyttää Huffman-koodausta, joka käyttää kahta kenttää. Ensimmäinen kenttä on staattinen ja sisältää pakatut versiot usein käytetyistä otsakkeista. Toinen kenttä on dynaaminen, ja siinä on muut pakatut otsakkeet session mukaan. (de Saxcé et al. 2015)

HTTP/2 on binäärinen protokolla. Aikaisemmat versiot olivat tekstipohjaisia, eli ne olivat suoraan ihmisen luettavissa. Binäärinen rakenne on huomattavasti tiiviimpi ja helpompi jäsentää (parse). (de Saxcé et al. 2015)

HOL-ongelman korjaamiseksi HTTP/2 esittelee vuot, jossa jokainen asiakkaan pyyntö saa oman vuon. Nämä vuot sitten kanavoidaan yhteen TCP-yhteyteen. Näin pyynnöt eivät vaikuta toisiinsa ja palvelin pystyy samanaikaisesti vastaamaan niihin. HOL-ongelman korjauksen lisäksi tämä pienentää tarvittavien TCP-yhteyksien määrää. (de Saxcé et al. 2015)

HTTP/2:n optimointi toimii eri tavalla kuin vanhoissa versioissa. Pelkkien HTTP-pyyntöjen määrän vähentämisen sijaan pitäisi muuttaa verkkosivujen välimuistin toimintaa.

Periaatteessa resurssien pitäminen pienissä paloissa johtaa siihen, että ne voidaan siirtää rinnakkain. Kuvassa 3.1 näkyy selvästi HTTP/1.1:n ja HTTP/2:n ero avattavien TCP-yhteyksien määrässä ja tavassa siirtää tietoa. Ylhäällä HTTP/1.1 avaa kolme TCP-yhteyttä saadakseen kaikki kolme palvelimen tarjoamaa resurssia. Alhaalla HTTP/2 siirtää kaiken rinnakkain yhden TCP-yhteyden sisällä. (Hodson 2015)



Kuva 3.1 HTTP/1.1 ja HTTP/2 välinen ero TCP-yhteyksissä. (Hodson 2015)

Palvelimen työntö (server push) on kyky ennakoida asiakkaan tekevän useita muita pyyntöjä palvelimelle ja lähettämään etukäteen hyödylliseksi katsomaansa dataa. Asiakkaan ei ole pakko hyväksyä työnnettyä dataa. Hyväksyttäessä data tallentuu asiakkaan selaimen välimuistiin (cache). (de Saxcé et al. 2015)

Tehokkuutta lisää myös pyyntöjen priorisoinnin salliminen. (Belshe et al. 2015) Tämä toteutetaan antamalla asiakkaan asettaa sijat samanaikaisille voille. Näin palvelin pystyisi lähettämään tärkeämpiä tietoja ensin (esimerkiksi kuvien sijaan) ja näin teoriassa lyhentämään sivujen latausaikaa. Siten selain voi jäsentää tärkeät tiedot ensin ja lähettää omat pyynnöt referoiduille kuville. (de Saxcé et al. 2015)

3.1 HTTP/2-yhteyden luominen

Yhteyden muodostaminen HTTP/2:n kanssa voi tapahtua kolmella eri tavalla (Torii 2016):

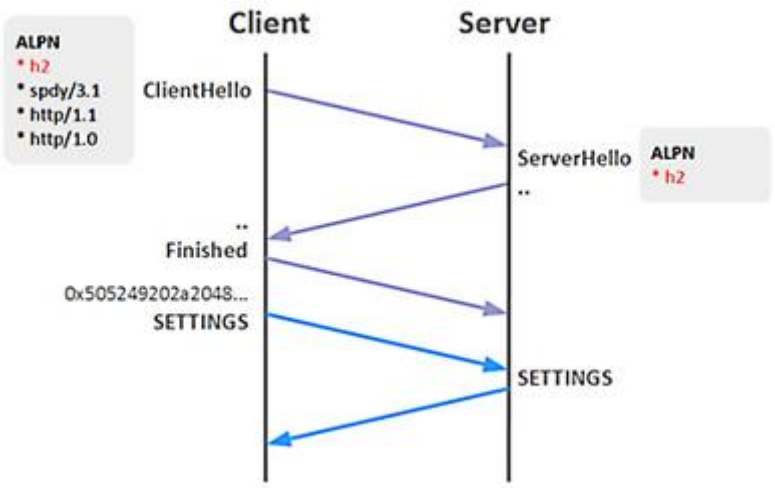
- Päivittämällä HTTP/1.1:stä.
- Neuvottelemalla TLS-salausprotokollan laajennuksella ALPN.
- Aloittamalla etukäteistiedolla HTTP/2-yhteydestä.

Suojaamattomalla yhteydellä asiakas tekee HTTP/1.1-pyynnön salaamattomana tekstinä, joka sisältää päivitysotsakekentän. Palvelin voi hyväksyä päivitetyn yhteyden lähettämällä viestin protokollan vaihdosta ja siirtyä käyttämään HTTP/2:ta. Palvelin voi myös ohittaa päivitysotsakekentän vastauksessa ja jatkaa HTTP/1.1:n käyttöä, jos tukea HTTP/2:lle ei ole. Protokollan vaihdon jälkeen asiakas voi alkaa lähettää HTTP/2-kehyksiä (frame). (Belshe et al. 2015)

Suojatulla yhteydellä asiakas tekee pyynnön käyttämällä TLS-salausprotokollan laajennusta TLS-ALPN (Belshe et al. 2015). Tässä asiakas lähettää palvelimelle listan mahdollisista yhteysprotokollista ja palvelin valitsee yhden. Kun HTTP/2-palvelin löytää h2-merkinnän, joka tarkoittaa HTTP/2-yhteyttä, niin palvelin ilmoittaa asiakkaalle aloittavansa HTTP/2-yhteyden. Jos tätä merkintää ei löydy listasta, niin palvelin aloittaa yhteyden HTTP/1.1:n kanssa. (Torii 2016)

Jos palvelimen tiedetään tukevan HTTP/2-yhteyttä, niin asiakas voi ohittaa päivitysotsakkeiden käytön ja yhdistää suoraan lähettämällä yhteyden johdannon (connection preface). Yhteyden johdanto on 24 oktettia pitkä merkkijono, joka lähetetään yhteyden avaamisen yhteydessä. (Undertow 2015)

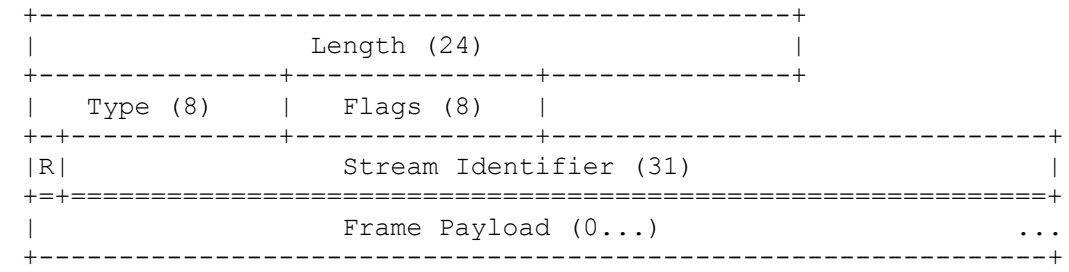
Ensimmäiset pakolliset viestit yhteyden avaamisessa ovat yhteyden johdanto ja asetuskehykset edestakaisin yhteyden alkuasetuksiksi. Asetukset voivat myös olla tyhjiä, mutta ne lähetetään aina. (Torii 2016) (Undertow 2015) Kuvassa 3.2 esitellään yhteyden avaaminen suojatulla yhteydellä:



Kuva 3.2 HTTP/2-yhteyden muodostus TLS-ALPN suojatulla yhteydellä. (Torii 2016)

3.2 Kehys

Yhteyden muodostamisen jälkeen asiakas ja palvelin voivat aloittaa vaihtamaan kehyksiä. Seuraavaksi käydään läpi kehys kenttä kerrallaan kuvan 3.3 pohjalta.



Kuva 3.3 Kehyksen pohja. (Belshe et al. 2015)

Pituuskenttä (length) kertoo kehyksen koon 24-bittisenä. Kentän koko on rajoitettu ja sitä voi kasvattaa pyytämällä palvelimelta suurennosta. Tyyppikenttä (type) määrittää kehyksen formaatin ja semantiikan. Sovelluksen täytyy ohittaa kaikki kehykset, joiden tyyppi on tuntematon. Lippukenttä (flags) on varattu totuusarvomuuttujille, jotka kuuluvat kehyksen tyypille. Varattu kenttä (R) on semantiikaltaan määrittelemätön kenttä. Se täytyy jättää asettamatta lähettäessä ja se täytyy ohittaa vastaanottaessa. Vuontunnistekenttä (steam identifier) sisältää vuon tunnisteen. Kehyksen sisältö (frame payload) riippuu täysin kehyksen tyypistä. (Belshe et al. 2015) Taulukossa 3.1 on esitelty kehyksen tyyppikentät.

Taulukko 3.1 Kehyksen tyyppikentät. (Torii 2016)

Tyyppi	Kehyksen tyyppi	Kuvaus
0	DATA	Pyynnön/vastauksen sisältöosa
1	HEADERS	Pyynnön/vastauksen otsakeosa
2	PRIORITY	Prioriteetin asettaminen voille (vain asiakas)
3	RST_STREAM	Vuon lopettaminen
4	SETTINGS	Yhteysasetuksien vaihto
5	PUSH_PROMISE	Ilmoitus työnnöstä (vain palvelin)
6	PING	Yhteyden olemassaolon tarkistus
7	GOAWAY	Yhteyden katkaisu
8	WINDOW_UPDATE	Vastaanottoikkunan koon vaihto
9	CONTINUATION	Suuren kehyksen pala

Kehyksen maksimikoko vaihtelee 2^14 (16,384) ja 2^24-1 (16,777,215) oktetin välillä ja kaikkien sovellusten täytyy pystyä käsittelemään vähintään 2^14+9 oktetin kokoinen kehys. 9 uutta oktettia tulee kehyksen otsakekentästä. Päätepisteiden ei ole pakko käyttää koko kehyksen tilaa hyväkseen ja aikaherkkien kehyksien kanssa se voi vaikuttaa suorituskykyyn. (Belshe et al. 2015)

Yksi kehyksen sisältötyyppi on otsake. Otsake sisältää nimen yhdellä tai useammalla siihen kuuluvalla arvolla. Sitä käytetään HTTP-pyynnöissä ja vastauksissa, sekä palvelimen työntöominaisuudessa. (Belshe et al. 2015)

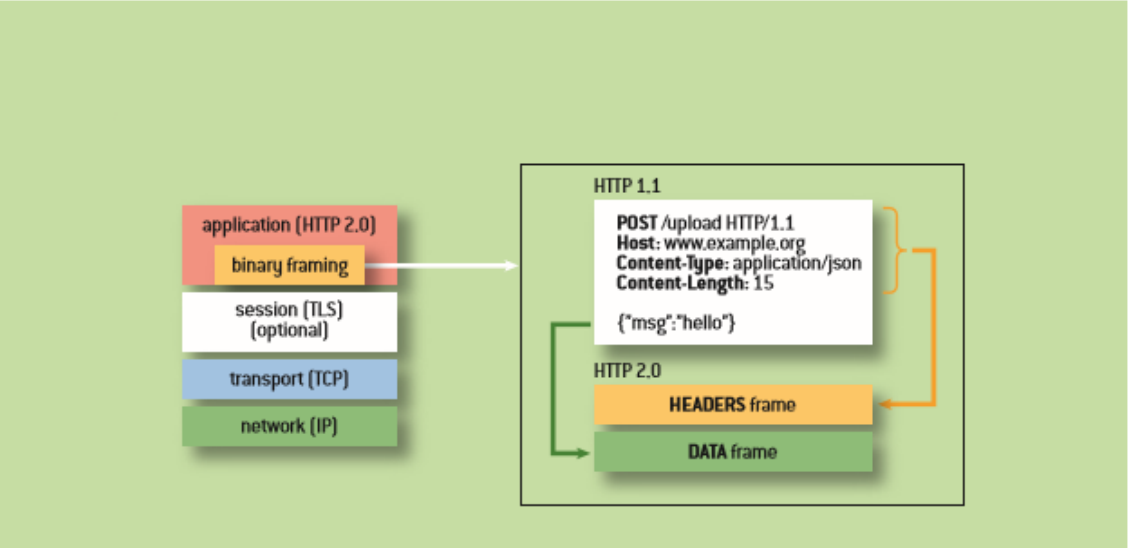
Otsakkeen pakkaus ja purku on tehokkuuden kannalta tärkeä toiminto. Pakkaukseen otsakkeet kootaan otsakelistoiksi, jotka ovat nollan tai useamman otsakekentän kokoisia. Tämä lista tiivistetään lähettäessä otsakepalaksi. Sen jälkeen se jaetaan osiin ja lähetetään kehyksen sisältönä. Näin otsakekehyksen sisältö on kaikki otsakkeet koottuna yhteen. Vastaanottava osapuoli kokoaa otsakelistan ketjuttamalla sen yhteen ja purkaa sen. Täydellisessä otsakelistassa on aina lippu, joka päättää otsakkeet, tai jos lippua ei ole, tulee jatkossa lisää otsakekehyksiä. (Belshe et al. 2015)

Otsakkeen tiivistys on tilallinen toiminta. Vain yhtä tiivistys- ja purkutilaa käytetään koko yhteyden ajan. Otsakepalat täytyy lähettää peräkkäisissä kehyksissä. (Belshe et al. 2015)

Muut tyyppikentät esitellään seuraavaksi. PRIORITY-kehyksellä asiakas voi asettaa tärkeysarvopainon vuolle. Tämä mahdollistaa asiakkaan kertoa, että miten se haluaa resursseja käytettävän rinnakkaisiin voihinsa. Tämä on tärkeä ominaisuus varsinkin jos resursseja on rajallisesti käytettävissä. RST_SREAM-kehys on varattu vuon lopettamiseen. Sen avulla vuo päätetään. SETTINGS-kehys määrittelee parametrit päätepisteiden kommunikointiin, kuten asetukset ja rajoitukset. Kenttää käytetään myös asetuksien vastaanottamisesta ilmoittamiseen. PUSH_PROMISE-kehyksen avulla palvelin pystyy ilmoittamaan asiakkaalle alkavasta työnnöstä. Tämä tapahtuu vastaamalla näillä lupauskentillä ennen alkuperäistä vastausta, jolloin asiakas tietää palvelimen työnnöstä. Näin asiakas ei pyydä työnnettävää dataa erikseen. Kehyksestä on myöhemmin kuva 3.10. PING on mekanismi, jolla voidaan mitata pienin kiertoaika lähettäjältä. PING-kehystä käytetään myös tunnistamaan onko käyttämätön yhteys vielä toiminnassa. GOAWAY-kehystä käytetään ilmoittamaan yhteyden lopettamisesta. Kehyksen lähettäjä lupaa olla lähettämättä enempää tietoa. WINDOW_UPDATE:a käytetään muuttamaan vastaanottoikkunan kokoa ja on esitelty tarkemmin kuvassa 3.7. CONTINUATION-kehystä käytetään, kun kehyksestä tulee niin suuri, että sitä ei voi enää lähettää yhdessä palassa. (Belshe et al. 2015)

3.3 Vuo ja kanavointi

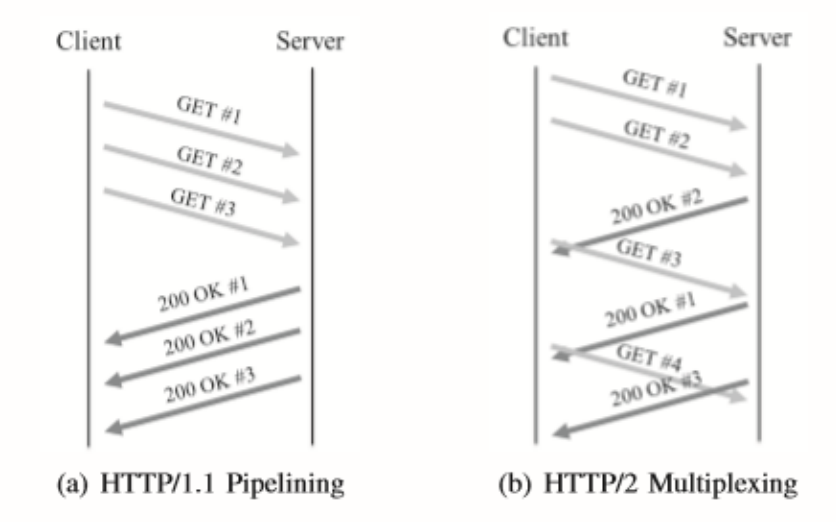
HTTP/2:n suorituskyvyn parantamisen olennainen osa on uusi binäärinen kehystyskerros. Se määrää miten HTTP-viestit kehystetään ja siirretään asiakkaan ja palvelimen välillä. HTTP:n semantiikkaan tämä ei vaikuta, mutta siirron ajaksi viestit on koodattu eri tavalla. Binäärinen kehystäminen on esitelty kuvassa 3.4. Kuvassa on vasemmalla eritelty kerrokset, joita käytetään. Kerrokset alhaalta ylöspäin ovat verkkokerros, kuljetuskerros, istuntokerros ja sovelluskerros. Ylimmän sovelluskerroksen HTTP/2 on avattu oikealle. Siinä esitetään, miten HTTP/1.1-viesti jaetaan kahteen eri kehykseen, lähetystiedot otsakekehykseen ja viestin sisältö datakehykseen. Tämä on seuraus HTTP/1.1-protokollan tavasta lähettää selvälukuista tekstiä, joka varmistaa, että yhdessä yhteydessä voidaan toimittaa vain yksi vastaus. HTTP/2:n binäärinen kehystyskerros poistaa tämän rajoituksen ja mahdollistaa pyyntöjen ja vastauksien kanavoinnin. (Grigorik 2013a)



Kuva 3.4 HTTP/2-sanoman binäärinen kehystys. (Grigorik 2013a)

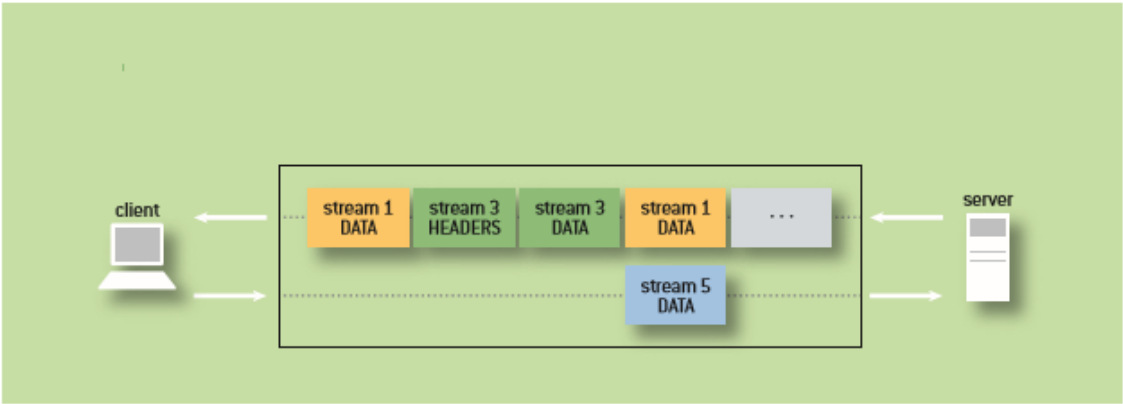
Vuo on itsenäinen ja kahdensuuntainen ketju kehyksiä, joita asiakas ja palvelin vaihtavat keskenään HTTP/2-yhteyden sisällä. Yhden yhteyden sisällä voi olla useita yhdenaikaisesti avoinna olevia voita. Molemmat, asiakas ja palvelin, voivat luoda uuden vuon ja käyttää sitä yhdensuuntaisesti tai jakaen. Molemmat päätepisteet voivat sulkea vuon. Vuon sisällä on tärkeää lähettää kehykset järjestyksessä, sillä ne käsitellään saapumisjärjestyksessä. Vuolle annetaan kokonaislukuna tunniste ja vuon aloittava päätepiste antaa sen. (Belshe et al. 2015)

Kuvassa 3.5 annetaan esimerkki kanavoinnin ja ketjutuksen erosta. Kuvan 3.5 kohdassa (a) HTTP/1.1-ketjutus uudelleenkäyttää TCP-yhteyttä vähentääkseen viivettä. Tässäkin on vielä ongelmia, kuten HOL. Ratkaistakseen tämän ongelman HTTP/2 lähettää useita pyyntöjä yhdessä TCP-yhteydessä, kuten kuvassa 3.5 kohdassa (b) näkyy. Kun yhteys on luotu, niin HTTP/2 ylläpitää yhteyttä, kunnes viestintä on tarpeetonta. Yhden yhteyden sisällä HTTP/2 luo monia itsenäisiä voita. Jokaisella vuolla on oma uniikki vuotunniste ja se toteuttaa useita HTTP-pyyntöjä. Näin voiden välillä olevaa lähetysjärjestystä ei tarvitse ylläpitää.



Kuva 3.5 HTTP 1.1 ja HTTP/2 siirto-ominaisuudet. (Kim et al. 2015)

Kuvassa 3.6 selvennetään kanavoinnin toimintaa. Yhden yhteyden sisään luodaan useita eri voita ja annetaan niille oma tunniste. Näin dataa voidaan lähettää samanaikaisesti ja se osataan kasata molemmissa päissä. Kuvassa keltainen, vihreä ja sininen kuvastavat eri voita. (Grigorik 2013a)



Kuva 3.6 HTTP/2 kanavointi yhden TCP-yhteyden sisällä. (Grigorik 2013a)

3.4 Vuonhallinta

Vuonhallintaa (flow control) tarvitaan, koska voiden käyttö kanavoinnissa aiheuttaa kilpailua TCP-yhteyden käytöstä. Vuonhallinta varmistaa, että samassa yhteydessä olevat vuot eivät vaikuta toisiinsa tuhoisasti. Vuonhallintaa käytetään koko yhteyteen ja myös yksittäisiin voihin. (Belshe et al. 2015)

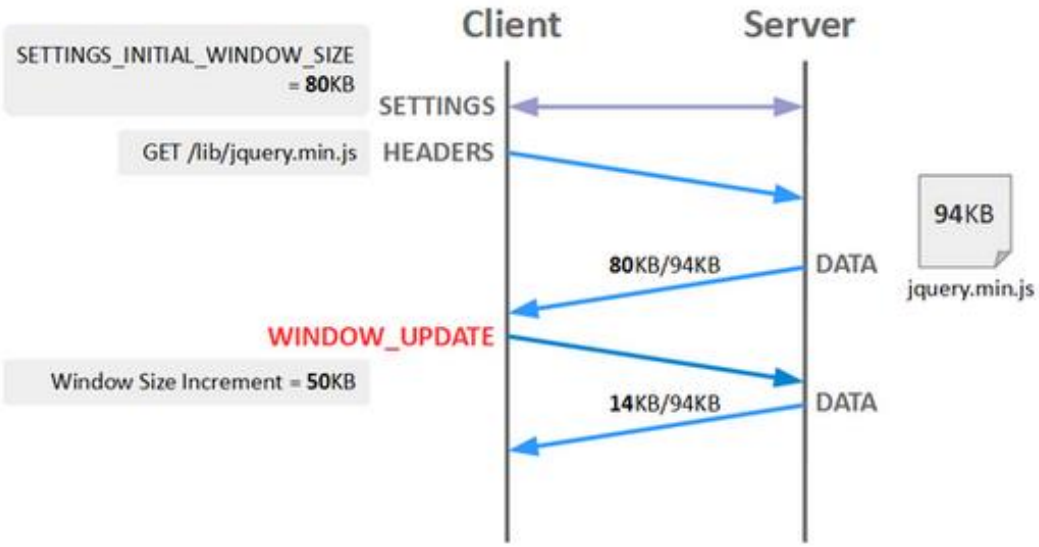
HTTP/2:n vuonhallinta pyrkii sallimaan useita vuonhallinta-algoritmeja ilman protokollamuutoksia. Vuonhallinta noudattaa seuraavia ominaisuuksia:

- Vuonhallinta on yhteyskohtainen. Hallinta on yksittäisten hyppyjen välinen, eikä koko yhteyden päästä päähän.

- Vastaanottajat kertovat kuinka paljon he ovat valmiita vastaanottamaan bittejä datakehyksessä vuossa ja koko yhteydessä. Määrän kertoo Window_Update-kehys.
- Vuonhallinta on suunnallinen ja vastaanottajalla on hallinta. Lähettäjän täytyy hyväksyä vastaanottajan asettamat rajat lähetykselle. Asiakkaan, palvelimet ja välittäjät mainostavat sallittuja vuon kokojaan itsenäisesti.
- Alkukoko vuonhallintakehykselle on 65 535 oktettia.
- Kehyksen tyyppi määrittää vuonhallinnan käytön. Vain datakehykset käyttävät vuonhallintaa. Näin kontrollikehykset eivät esty vuonhallinnan takia.
- Vuonhallintaa ei voi sammuttaa.
- HTTP/2 määrittää vain formaatin ja semantiikan Window_Update-kehykselle. Näin käyttäjät voivat valita haluamansa algoritmin.

Vuonhallinta on määritelty suojaamaan päätepisteitä, jotka toimivat rajallisilla resursseilla. Usein esimerkiksi asiakkailla on nopea latausnopeus, mutta hidas lähetysnopeus. Jos tätä suojausta ei tarvita, niin piste voi mainostaa Window_Update-kehykselle maksimikokoa. Tämä käytännössä sammuttaa vuonhallinnan. Vuonhallinnan käyttö voi olla hankalaa. Vastaanottajan on pakko lukea TCP:n vastaanottopuskuria, jotta kriittisiin kehyksiin, kuten Window_Update, voidaan reagoida ajallaan. Window_Update-kehyksen algoritmien kehitys jatkuu ja onkin hyvä alue tutkimukselle ja optimoinnille. (Belshe et al. 2015) (Grigorik, 2013a)

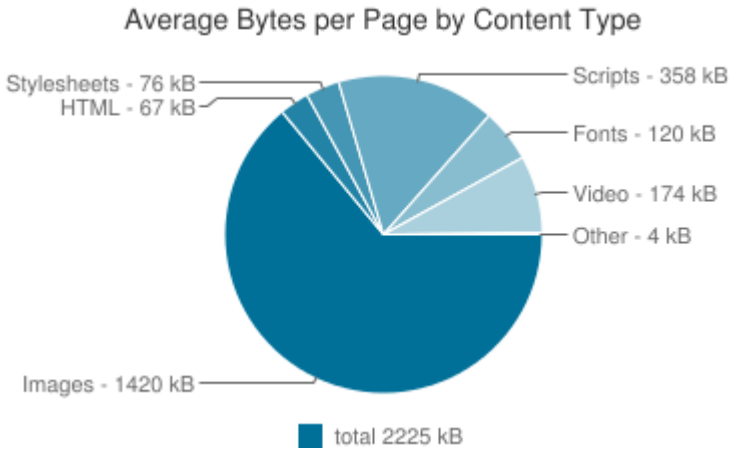
Kuvassa 3.7 on yksinkertaistettu esimerkki Window_Update-kehyksen toiminnasta. Alkuun kehyksellä on 80 kilotavun koko ja noudetaan 94 kilotavun tiedostoa. Palvelin lähettää 80 kilotavua tiedostosta ensin ja Window_Update kasvattaa lähetyskoon maksimia 50 kilotavua. Lähetettävää dataa on nyt vain enää 14 kilotavua, mutta olisi ollut mahdollista lähettää 130 kilotavua kerralla.



Kuva 3.7 Window_Update-kehyksen toiminta haettaessa dataa. (Torii 2016)

3.5 Vuon priorisointi

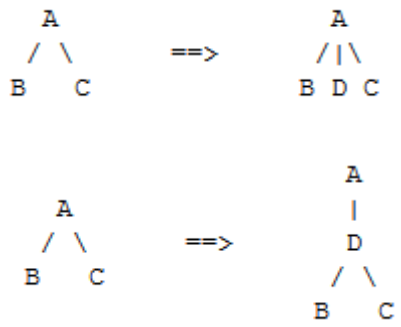
Kun HTTP-viesti voidaan jakaa yksittäisiin kehyksiin, niin niiden järjestystä voidaan optimoida. Tämä tapahtuu vapaaehtoisella 31 bitin tärkeysarvolla otsakekehyksessä, jossa 0 on tärkein vuo. Verkkosivua piirtäessä toiset tiedot ovat tärkeämpiä. HTML- dokumentti on kriittinen, koska se sisältää rakenteen ja referenssin muihin tietoihin. CSS-tyylitiedosto tarvitaan ennen kuin voidaan piirtää pikseleitä. Kuvia voidaan hakea pienemmällä prioriteetilla. Huomattavaa on, että prioriteetti on vain ehdotus, ja yhteyspistettä ei voi pakottaa käsittelemään samanaikaisia voita tietyssä järjestyksessä. (Grigorik, 2013a) Vuoden 2016 alussa keskimääräinen koko eri tiedostoille on esitelty kuvassa 3.8. Erilaiset kuvat ovatkin selvästi suurin osa verkkosivujen koosta.



Kuva 3.8 Verkkosivujen keskimääräinen sisältö. (Httparchive 2016)

Prioriteetti mahdollistaa päätepisteelle ilmoittaa, miten se haluaisi toisen päätepisteen käyttävän resursseja samanaikaisten voiden kanssa. Huomattavaa on, että prioriteettia voidaan käyttää valitsemaan voita lähetykseen, kun lähetykseen on käytettävissä rajallisia resursseja. Voita voidaan priorisoida myös merkitsemällä ne riippuvaiseksi toisen vuon valmistumisesta. Jokaiselle riippuvuudelle asetetaan myös suhteellinen painoarvo. Sitä käytetään selvittämään suhteellinen osa resursseista, jotka on annettu kyseisestä vuosta riippuville voille. Esimerkiksi painoarvolta 4 oleva vuo saa kolme kertaa enemmän resursseja kuin painoarvolla 12 oleva vuo. (Belshe et al. 2015)

Vuo, joka riippuu toisesta vuosta, on riippuvainen vuo. Toisista voista riippumaton vuo on vanhempi vuo. Kun vuo merkitään riippuvaiseksi toisesta vuosta, niin se merkitään uudeksi riippuvuudeksi vanhemmalle vuolle. Samasta vanhemmasta vuosta riippuvat vuot ovat yhdenvertaisia keskenään. Vuo voidaan merkitä myös poissulkevaksi riippuvuudeksi. Tällöin samasta vanhemmasta riippuvat muut vuot siirtyvät riippumaan poissulkevasta vuosta. Toiminta esitellään kuvassa 3.9. (Belshe et al. 2015)



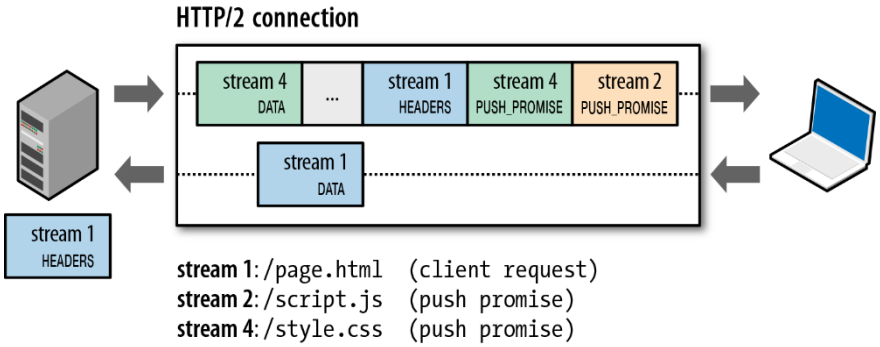
Kuva 3.9 Yllä uuden vuon D asettaminen riippuvaiseksi vuosta A. Alla vuon D asettaminen poissulkevaksi vuoksi. (Belshe et al. 2015)

3.6 Palvelimen työntö

Palvelimen työntö on HTTP/2:n uusi tehokas ominaisuus. Se avulla palvelin voi lähettää useita vastauksia yhteen pyyntöön. Alkuperäisen pyynnön vastauksen lisäksi palvelin voi siis lähettää lisää dataa asiakkaalle ilman, että asiakas pyytää jokaista resurssia erikseen. Kun asiakas pyytää esimerkiksi verkkosivua, joka koostuu useista resursseista, selain löytää kaikki resurssit ensimmäisestä palvelimen vastauksesta ja pyytää ne erikseen. Koska palvelin tietää jo ensimmäisestä pyynnöstä, että asiakas haluaa verkkosivun datan, niin palvelimen työnnössä lähetetään koko sivun data ilman lisäpyyntöjä. (Grigorik, 2013a)

HTTP-protokollan lisänä palvelimen työntö on uusi ominaisuus. Monet verkkosovellukset käyttävät sitä kuitenkin jo sisällyttämisenä (inlining). Verkkopalvelun tuottajat voivat sisällyttää resursseja itse verkkosivuun. Tämä data voi olla JS- ja CSS-koodia tai muita resursseja kuten kuvia ja ääntä. Jos sama data on sisällytetty usealle eri sivulle, on sisällytetty data noudettava joka kerta, jolloin sivun koko kasvaa. Näin sovellukset käytännössä työntävät dataa asiakkaalle. HTTP/2:n tapauksessa sisällyttäminen siirtyy pois sovelluksesta ja siirtyy suoraan protokollaan. Näin asiakas voi tallentaa työnnettävää dataa välimuistiin, hylätä sen, käyttää sitä useilla sivuilla ja palvelin voi priorisoida sitä. Käytännössä palvelimen työntö tekee sisällyttämisestä vanhentuneen. (Grigorik, 2013a)

Kuvassa 3.10 esitellään palvelimen työntöä. Asiakas pyytää html-sivua vuossa 1. Palvelin vastaa ilmoittamalla palvelimen työnnöstä luoden uusia voita 2 ja 4 Push_Promise-kehyksen avulla. Palvelin vastaa samalla alkuperäiseen pyyntöön vuossa 1 ja lähettää uusissa voissa JavaScriptin ja CSS-tyylitiedostot.



Kuva 3.10 Palvelimen työntö. (Grigorik, 2013b)

Push_Promise-kehystä käytetään ilmoittamaan päätepisteelle etukäteen voista, joita palvelin aikoo lähettää. (Belshe et al. 2015) Palvelimen on tärkeää luoda uudet vuot ja lähettää Push_Promise-ilmoitukset ennen ensimmäistä vastausta, jotta asiakas ei ehdi luomaan kaksoiskappaleita voista. (Grigorik, 2013b)

3.7 Otsakkeen pakkaus

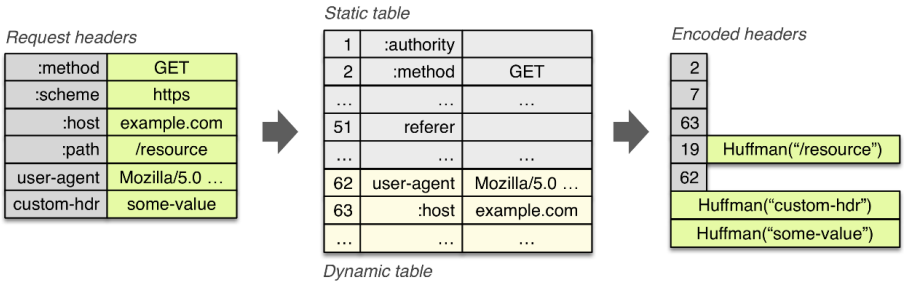
HTTP/1.1:n otsakekenttiä ei ole pakattu. Turhat kopiot otsakekentistä käyttävät kaistaa turhaan ja lisäten viivettä. SPDY pystyi pakkaamaan turhia kopioita, mutta sen tapa pakata osoittautui haavoittuvaiseksi hyökkäyksille. (Peon, Ruellan 2015)

HTTP/2:ssa käyttöön otettiin uusi HPACK-niminen tapa pakata otsakekentät. Se poistaa turhat otsakentät, rajoittaa haavoittuneisuutta tunnetuille hyökkäyksille ja sisältää rajoitetun muistiriippuvuuden käytettäväksi rajoitetuissa ympäristöissä. HPACK on tarkoituksenmukaisesti yksinkertainen ja joustamaton. Tämä vähentää toimimattomuuden ja turvallisuusasioiden riskiä. Muutoksia ei pystytä tekemään, joten HPACK voidaan vain vaihtaa kokonaan. (Peon, Ruellan 2015)

HPACK käyttää kahta yksinkertaista, mutta tehokasta tekniikkaa. Se sallii lähetettävien otsakekenttien pakkaamisen Huffmanin koodilla, joka pienentää lähetyskokoa. Lisäksi se vaatii asiakasta ja palvelinta ylläpitämään listaa aikaisemmin nähdyistä otsakekentistä. Tätä listaa käytetään tehokkaasti koodaamaan aikaisemmin lähetettyjä arvoja. (Grigorik, 2013b)

HPACK säilyttää otsakekentän järjestyksen otsakelistassa. Järjestyksen ylläpito on tärkeää pakkauksen ja purun takia. Otsakelistan purkua varten täytyy ylläpitää dynaamista taulukkoa referenssinä. Kaksisuuntaisen yhteyden (kuten HTTP) kanssa koodaus- ja purkutaulukot ovat itsenäisiä. Päätepisteet siis pitävät omia dynaamisia taulukoitaan. Järjestelmässä on myös oma staattinen taulukko, joka sisältää ennalta määritetyn listan otsakekenttiä. (Peon, Ruellan 2015)

Staattisessa taulukossa on 61 kohtaa. Dynaaminen taulukko on aluksi tyhjä. Lisäyksiä tehdään, kun otsakelistoja puretaan. Dynaaminen ja staattinen taulukko yhdistetään yhdeksi hakemistoksi niin, että dynaamisen listan kohdille annetaan arvoja 62 eteenpäin. (Peon, Ruellan 2015) Kuvassa 3.11 annetaan esimerkki toiminnasta. Kuvassa on vasemmalla pyyntö, jonka sisältö katsotaan kuvan keskellä olevasta hakemistosta pakattavaksi. Oikealla on pakattu otsake. Näin metodi GET muuttuu pakkauksessa vain numeroksi 2 staattisesta taulukosta. Dynaamisessa osassa taulukkoa on jo kohta user-agent, joten se voidaan pakata vain numeroksi 62.



Kuva 3.11 HPACK-otsakkeen pakkaus. (Grigorik, 2013b)

Dynaamista taulukkoa täytyy myös hallita. Muistivaatimuksien takia sen kokoa on rajoitettu. HPACK:ia käyttävien protokollat määrittelevät maksimikoon dynaamiselle taulukolle itse. Dynaamisen taulukon maksimikokoa voidaan myös muuttaa kesken yhteyden. Dynaaminen taulukko voidaan tyhjentää asettamalla taulukon maksimikooksi 0. Jos uusia arvoja lisätään taulukon ollessa jo täynnä, niin vanhoja tietoja poistetaan kunnes uusi tieto mahtuu taulukkoon. (Peon, Ruellan 2015)

3.8 Virheenhallinta

HTTP/2-kehystys sallii kahden tyyppisiä virheitä. Yhteysvirhe on koko yhteyden tuhoava virhe ja vuovirhe on vain yksittäisessä vuossa oleva virhe. (Belshe et al. 2015)

Päätepisteen, joka kohtaa yhteysvirheen, pitäisi lähettää GOAWAY-kehys yhteyden katkaisuun. GOAWAY:n täytyy sisältää viimeisen vuotunnisteen siitä vuosta, joka on viimeiseksi vastaanotettu onnistuneesti. Kehyksen lähetyksen jälkeen päätepisteen täytyy sulkea TCP-yhteys. On mahdollista, että toinen päätepiste ei luotettavasti vastaanota GOAWAY-kehystä. Se onkin vain paras mahdollinen yritys kertoa yhteyden lopetuksen syystä. Yhteyden katkaisu voi johtaa datan menetykseen. Päätepiste voi lopettaa yhteyden koska tahansa. Päätepiste voi myös käsitellä vuovirhettä yhteysvirheenä. (Belshe et al. 2015)

Vuovirhe vaikuttaa vain tiettyyn vuohon. Päätepiste, joka kohtaa vuovirheen, lähettää RST_STREAM-kehyksen. Kehys sisältää vuotunnisteen vuosta, jossa virhe tapahtui, sekä virheen tyypin. RST_STREAM on viimeinen kehys, jonka päätepiste voi lähettää

vuossa, mutta päätepisteen täytyy silti olla valmiina vastaanottamaan jo lähetettyjä kehyksiä. Vastaanotetut kehykset voidaan ohittaa, paitsi siinä tapauksessa, että ne muuttavat yhteyden tilaa, kuten esimerkiksi vuonhallinnan muutoksissa. (Belshe et al. 2015)

3.9 Turvallisuus

Internet mahdollistaa yrityksien ja yksilöiden käyttämään hyväkseen yhdistynyttä maailmaa. Jotkut tahot pyrkivät Internetissä kuitenkin toiminnallaan häiritsemään normaalia toimintaa ja saavuttamaan epäreilua etua. Yritysten palvelimien ja linkkien kimppuun hyökätään rahallisen hyödyn tavoittamistarkoituksessa. (Adi et al. 2015) Tässä aliluvussa esitellään erilaisia mahdollisia hyökkäyksiä HTTP/2-protokollaa vastaan ja perehdytään siihen, kuinka riittävä suojaus protokollassa on.

HTTP/2:n semantiikka asiakkaan ja palvelimen välillä on pysynyt samana kuin HTTP/1.1:n kanssa, mutta HTTP/2 vaatii enemmän laskentatehoa. Tästä seuraa, että HTTP/2 on HTTP/1.1:tä haavoittuvaisempi palvelunestohyökkäyksille (DoS). (Adi et al. 2015) HTTP/2 ja HTTP/1.1 käyttävät samaa tapaa määritellä palvelimen luotettavuus. Tämä luottaa paikalliseen nimentunnistukseen ”http” URI:n kanssa ja luotettujen palvelimien tunnisteeseen ”https” kanssa. (Belshe et al. 2015)

Palvelunestohyökkäyksillä voidaan tukehduttaa Internetissä kiinnioleva laite suurella määrällä liikennettä tai hyväksikäyttämällä haavoittuvuutta laitteen ohjelmistossa. Hyökkäystä voidaan kasvattaa käyttämällä tietokoneiden verkkoa aiheuttamaan liikenteen määrän myrsky. Verkon käyttäminen tunnetaan hajautettuna palvelunestohyökkäyksenä (DDoS). Tällaiset hyökkäykset ovat helposti huomattavia ja tavoitteena on lamauttaa. Jos halutaan pysyä huomaamattomana, niin hidas DoS-hyökkäys on vaihtoehto. Hitaassa DoS-hyökkäyksessä hyväksikäytetään verkkoprotokollan haavoittuvuuksia. (Adi et al. 2015) Tämä on mahdollista käyttämällä resursseja turhaan, jos päätepiste ei monitoroi toimintaansa tarkasti. HTTP/2-yhteys voi vaatia enemmän resursseja toimiakseen kuin HTTP/1.1. Eri kehyksien ylikäyttöä voidaan käyttää hyökkäykseen. Asiakas, joka hyväksyy PUSH_PROMISE-kehyksiä, pitää rajoittaa varattujen voiden määrää. Liiallista käyttöä voidaan pitää vuovirheenä. SETTINGS-kehyksiä voidaan käyttää vaatimaan ylimääräistä käsittelyaikaa, jos samoja asetuksia vaihdetaan jatkuvasti. Myös WINDOW_UPDATE- ja PRIORITY-kehyksiä voidaan käyttää resurssien hukkaamiseen. (Belshe et al. 2015) Tutkittaessa hitaan DoS-hyökkäyksen tehokkuutta HTTP/2-palvelimia vastaan ei pystytty lisäämään edes yhden millisekunnin viivettä palvelimen toimintaan (Adi et al. 2015).

Protokollien välisessä hyökkäyksessä pyritään aloittamaan keskustelu palvelimelle, joka ymmärtää eri protokollaa. Hyökkääjä voi saada keskustelun näyttämään oikealta toisessa protokollassa. Tätä voidaan hyväksikäyttää huonosti suojattujen palvelimien kanssa yksityisverkoissa. HTTP/2:n tapauksessa TLS-kättely ALPN-tunnisteella voidaan pitää riit-

tävänä suojauksena tällaisia hyökkäyksiä vastaan. ALPN kertoo, että palvelin haluaa jatkaa HTTP/2:n kanssa, mikä estää toiset TLS-pohjaiset protokollat. TLS-salaus tekee vaikeaksi hallita dataa, jota voitaisiin käyttää protokollien välisessä hyökkäyksessä selväkielistä protokollaa vastaan. Selvätekstisessä versiossa HTTP/2-protokollasta on minimaalinen suojaus tällaista hyökkäystä vastaan. Yhteyden aloituksessa on merkkijono, jonka tarkoituksena on sekoittaa HTTP/1.1-palvelimia, mutta ei suojausta muilta protokollilta. (Belshe et al. 2015)

Välittäjän kapselointihyökkäyksessä HTTP/2-otsakentän koodaus mahdollistaa nimien määritelmän, jotka eivät ole sallittuja HTTP/1.1:n kanssa. Pyynnöt ja vastaukset epäkelvon otsakekenttänimen kanssa täytyy käsitellä epämuotoisena. Näin välittäjä eivät voi kääntää HTTP/2-pyyntöä epäkelvon nimen kanssa HTTP/1.1-viestiksi. HTTP/2 mahdollistaa epäkelpoja otsakentän arvoja. Hyökkääjä voisi hyväksikäyttää tätä, jos ne käännettäisiin sanatarkasti. Näin kaikki pyynnöt, jotka sisältävät ei-sallittuja merkkejä täytyy käsitellä epämuotoisena. (Belshe et al. 2015)

HTTP/2 mahdollistaa liikenteen seurauksella tarkkailijalle mahdollisuuden yksilöidä yksittäinen asiakas tai palvelin. Tämä tapahtuu seuraamalla toimintaa, kuten asetuksien arvoja, vuonhallintaikkunoiden kokoja, prioriteettien asetuksia voissa ja eri muutettavien asetuksien toimintaa. Jos tämä aiheuttaa tunnistettavia poikkeuksia käyttäytymisessä, niitä voitaisiin käyttää tunnistamaan tietty asiakas. HTTP/2:n tapa käyttää yhtä TCP-yhteyttä mahdollistaa käyttäjän aktiviteetin tunnistamiseen palvelimelle. Joissain tilanteissa myös välitöntä vastausta vaativat PING- ja SETTINGS-kehykset saattavat vaarantaa käyttäjän yksityisyyden. (Belshe et al. 2015)

4. HTTP/2:N TEHOKKUUDEN TUTKIMUS

Tässä luvussa tarkastellaan HTTP/2:sta tehtyjä testejä ja verrataan sen tehokkuutta HTTP/1.1:tä ja SPDY:tä vastaan.

4.1 HTTP/1.1 vastaan HTTP/2

De Saxcé et al. (2015) tutkimuksessa selvitetään HTTP/2:n nopeutta ladata verkkosivu HTTP/1.1:tä vastaan. Heidän mielestään se on paras vertailukohta käyttäjän ja palvelun-tarjoajan kannalta.

Testipohjana toimii kaksi eri kokoonpanoa. Ensimmäisessä on lähiverkossa kaksi konetta ja toisessa Internet-laajakaistayhteys palvelimelle. Molemmissa kokoonpanoissa on asiakasselain ja palvelin, jotka tukevat molempia HTTP:n versioita, sekä testisisältö. Koska testin tekohetkellä joulukuussa 2014 selaimet tukivat HTTP/2:ta vain TLS-suojatun yhteyden yli, käytettiin testeissä Chromium-selaimen versiota, josta kytkettiin suojaus pois päältä. Testin tekijöiden mielestä TLS-suojauksen käyttö lisää viivettä merkittävällä tavalla ja selvittääkseen itse protokollien eroja testit tehtiin suojaamattomina.

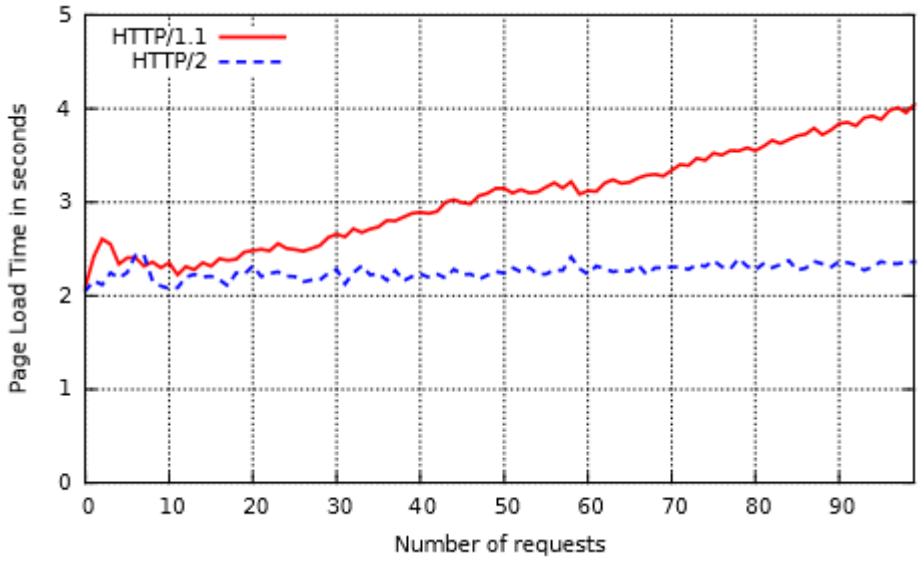
Testien sisältönä käytettiin kloonattuja verkkosivuja. Verkkosivuiksi valittiin Alexa-palvelun 15 suosituinta sivustoa. Kuvassa 4.1 esitellään sivustot ja niiden sisältämä data HTML:n, CSS:n, JS:n ja kuvien määrinä.

	Website	HTML	CSS	JS	Images
1	Google	2	1	5	4
2	Youtube	7	6	18	45
3	Wikipedia	3	2	5	5
4	Yahoo	4	2	8	38
5	Baidu	9	0	13	16
6	Amazon	4	28	40	45
7	Taobao	7	11	55	93
8	QQ	21	1	29	132
9	Weibo	7	4	11	20
10	Tmall	8	6	8	69
11	Ebay	9	5	8	76
12	Hao123	7	2	20	57
13	Yandex	2	0	6	12
14	Sohu	73	4	93	219
15	Bing	4	0	13	6

Kuva 4.1 Testeihin valitut verkkosivut ja niiden sisältö. (de Saxcé et al. 2015)

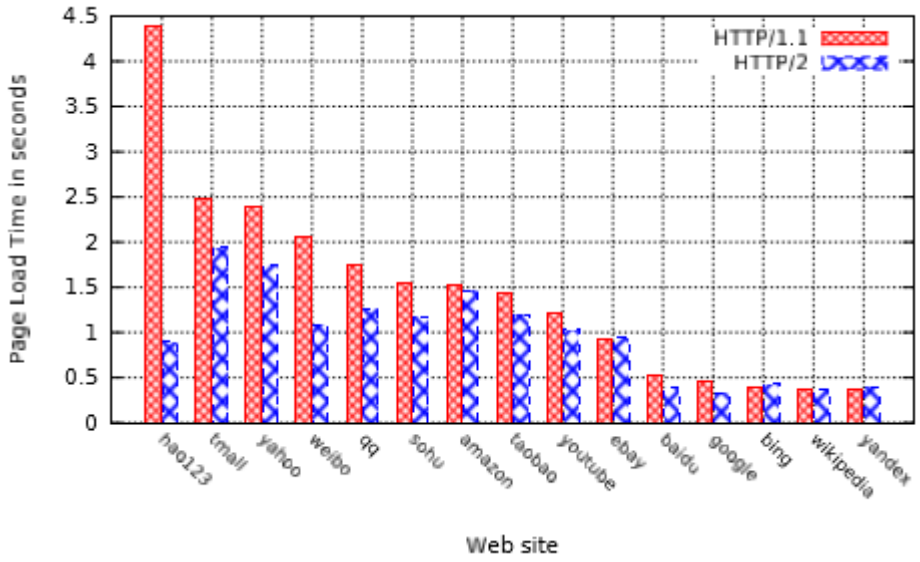
Testit sisältävät myös yksinkertaistuksia kokoonpanosta johtuen. Testeissä ei käytetty domainin palastelua. Niissä käytettiin suoraan IP-osoitteita eikä domainin nimitunnistusta. 3G-modeemi oli kytkettynä tietokoneeseen, joten ei tullut CPU-rajoituksia kuten tehottomammassa älypuhelimessa.

Lähiverkossa kokeiltiin erillisellä testisivulla yhden megatavun kuvan hakua. Kuva haettiin kokonaisena ja jakamalla se pienempiin osiin. Osina käytettiin maksimissaan sataa ja viiveenä 100 ms. Kuvassa 4.2 nähdään selvästi HTTP/1.1-ongelmana oleva HOL, kun sivun latausaika kasvaa pyyntöjen kasvaessa ja HTTP/2-latausaika pysyy vakaana eri pyyntömäärillä. Jo tällä yksinkertaisella testillä voidaan päätellä, että monimutkaisemmat sivut suurilla hakumäärillä hyötyvät HTTP/2:een siirtymisestä eniten.



Kuva 4.2 Latausaika eri pyyntöjen määrillä. 100 ms viive. (de Saxcé et al. 2015)

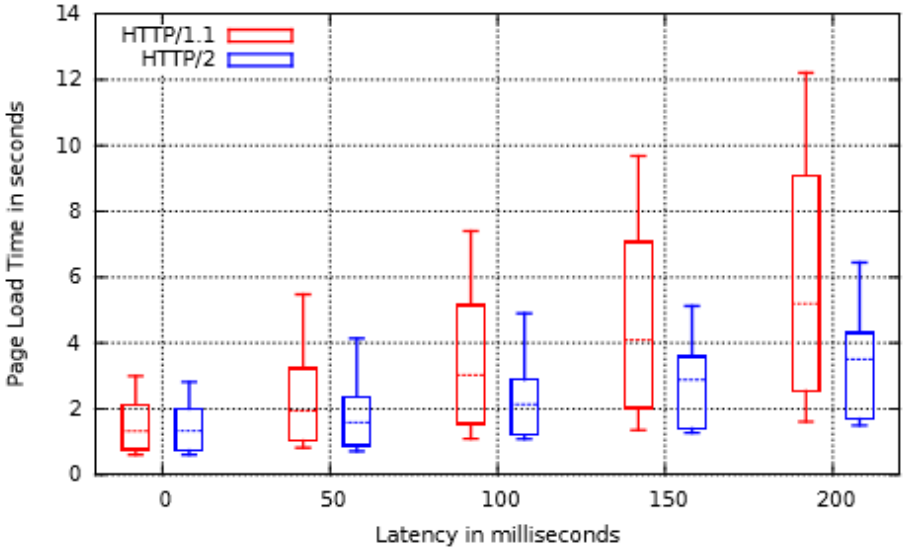
Varsinaisten verkkosivujen testissä laajakaistayhteydellä nähdään HTTP/2:n hyöty. Verkkosivujen latausajat on esitelty kuvassa 4.3. Keskiarvoinen etu HTTP/2:n kanssa oli noin 20 %. Suuremmat sivut nopeutuivat enemmän ja hyöty yksinkertaisilla sivuilla oli lähes olematon. Hao123-sivu on poikkeavuus. Scripti hidasti HTTP/1.1:n latautumista



Kuva 4.3 Laajakaista 50 ms viiveellä. (de Saxcé et al. 2015)

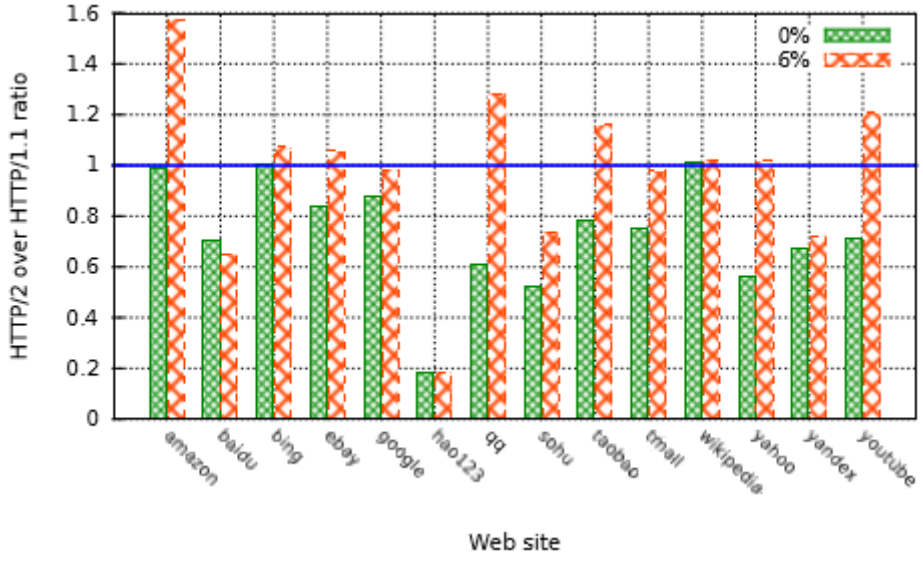
Sama testi ajettiin myös 3G-modeemilla ja tulokset olivat hyvin samankaltaisia antaen noin 20 % hyödyn HTTP/2:n kanssa. Testiin asetettiin 400 ms viive, mutta ei pakettien katoamista. Pakettihäviön odotetaan aiheuttavan ongelmia HTTP/2:n kanssa. Seuraavaksi tehtiin lähiverkkotestejä, jotta voitiin hallita työkaluilla viiveen määrän lisäksi myös pakettihäviön määrää.

Tutkittiin myös viiveen lisäämisen vaikutusta protokolliin. Otettiin minimi, maksimi ja mediaani latausaika. Kuvassa 4.4 nähdään selvästi HTTP/2:n reagoivan erittäin hyvin viiveeseen.



Kuva 4.4 Viiveen vaikutus protokolliin. HTTP/1.1 vasemmalla punaisella ja HTTP/2 oikealla sinisellä. (de Saxcé et al. 2015)

Seuraava kiinnostava asia oli matkapuhelinverkon pakettihäviö. Kuvassa 4.5 nähdään latausajat ilman pakettihäviötä ja 6 % pakettihäviön kanssa. Testi on osoitettu suhteella, jossa alle yhden olevissa tuloksissa HTTP/2 on nopeampi ja yli yhden olevissa tuloksissa HTTP/1.1 on nopeampi. Tämä osoittaa huonoa toimintaa pakettihäviön kanssa. Se voidaan selittää sillä, että HTTP/2 käyttää vain yhtä TCP-yhteyttä, joten pakettihäviö vaikuttaa yhtäaikaisesti jokaiseen vuohon. HTTP/1.1:n kanssa on auki useita TCP-yhteyksiä, mikä vähentää pakettihäviön aiheuttamaa viivettä.



Kuva 4.5 Pakettihäviön vaikutus protokolliin kiinteällä viiveellä. (de Saxcé et al. 2015)

4.2 Dynaaminen mukautuva suoratoisto

Cherif et al. (2015) tutki HTTP/2:n käyttöä dynaamisessa mukautuvassa suoratoistossa (DASH). DASH on yleisesti käytössä oleva videoiden suoratoistomenetelmä. Tutkimuksessa tarkasteltiin HTTP/2:n toimintaa viiveen vähentämisessä DASH-toiston aloittamiseen käyttämällä ominaisuutta nopea aloitus (fast start).

Tutkimuksessa esitellään uusi tapa arvioida kaistanleveyttä. Arviotapa on nimeltään Web Socket (WS). WS mahdollistaa kaksisuuntaisen kanavan luomisen yhteen TCP-yhteyteen asiakkaan ja palvelimen välille. Näin viestejä voidaan lähettää hyvin pienellä viiveellä. HTTP/2 mahdollistaa muitakin protokollia kulkemaan saman yhteyden sisällä. Näin asiakas ja palvelin voivat käyttää yhtä TCP-yhteyttä kuljettamaan sekä HTTP- että WS-viestejä. Tutkimuksessa verrataan myös WS:n toimintaa omalla TCP-yhteydellä verrattuna yhdistettyyn TCP-yhteyteen.

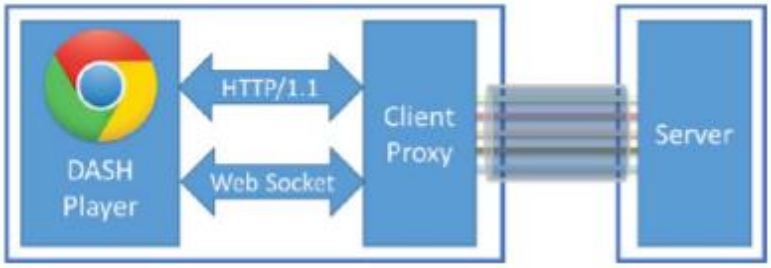
Normaalilla DASH-asiakkaalla video ei ala toistua ennen kuin puskuriin on ladattu ennalta määrätty määrä videota. Puskurin täyttymiseen HTTP/1.1:n kanssa täytyy lähettää pyyntö jokaiseen videon osaan erikseen ja odottaa palvelimen vastausta. Nopeuttamaan tätä toimintaa ehdotetaan DASH:n nopeaa aloitusta. Tämä tapahtuisi työntämällä video-

osia ilman asiakkaan pyyntöjä. Jos asiakas ei halua työnnettyä videon osaa, niin se voi kieltäytyä siitä. Asiakkailla ei välttämättä ole tietoa, koska osia työnnetään, mikä tekee kaistanleveyden arvioinnista vaikeaa. WS-yhteyden käyttö HTTP/2:n yli korjaa tämän ongelman.

Asiakassovellus ei tiedä työnnetyn datan kokoa tai bittinopeutta. Käyttämällä WS-ohjausdataa se voi laskea datan läpisyötön. WS-data onkin vain merkintä jokaisen videon osan alussa ja lopussa. Merkintöjen välisestä ajasta saadaan helposti läpisyöttö. Normaalilla viestinnällä lähetyskyky kasvaa hiljalleen, kun taas WS:n kanssa läpisyöttö voidaan laskea ja kasvattaa lähetyskyky suoraan maksimiinsa. Kuvassa 4.6 esitellään WS-viestintä. WS ilmoittaa dataa lähettäessä sekä lähetyksen aloituksen että lopetuksen. Näin asiakassovellus saa viestit lähetysjärjestyksessä.

Kuva 4.6 WS-ohjausdatan lähetys. Aloitus- ja lopetusmerkinnät videon osan ympärillä. (Cherif et al. 2015)

Testiympäristö löytyy kuvasta 4.7. Se koostuu kolmesta osasta. Oikealla on verkkopalvelin, keskellä asiakaskone ja vasemmalla verkkoasiakas, joka on myös asiakaskoneessa.



Kuva 4.7 Testiympäristö. (Cherif et al. 2015)

Palvelin pystyy työntämään video-osia staattisten sääntöjen mukaan. Yksi testatuista säännöistä on media presentation description (MPD) -tiedoston vastaanottaminen. MPD sisältää videon osien tiedot ja metadatan. Sen jälkeen palvelin työntää aloitustiedot ja video-osat, kunnes asiakas ilmoittaa videon alkaneen toistua. Palvelin aloittaa työnnön uudelleen, kun asiakas ilmoittaa puskurin alkavan tyhjentyä. Palvelin käyttää HTTP/2-yhteyttä lähettämään ja vastaanottamaan WS-viestejä.

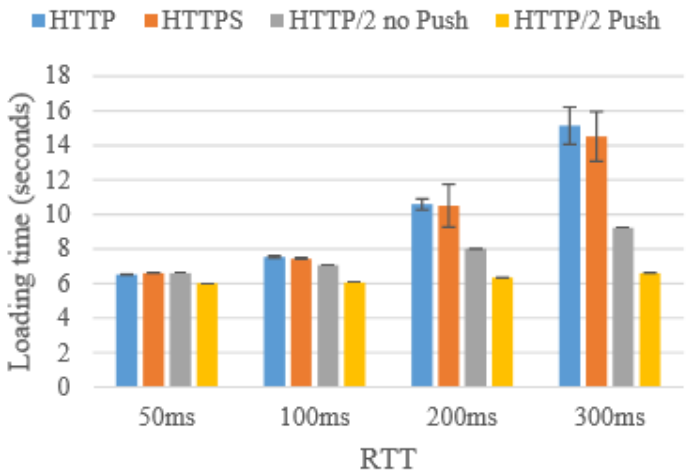
Asiakaskone tukee HTTP/2:ta, WS:ää ja HTTP/1.1:tä. Asiakaskone pystyy tallentamaan palvelimen työntämiä videon paloja välimuistiin. Asiakaskone ohjaa verkkoasiakkaan viestit palvelimelle. Arvioimaan HTTP/1.1:n tehokkuutta asiakaskone käyttää kuutta samanaikaista TCP-yhteyttä, kuten Google Chrome ja monet selaimet tekevät. Koska

HTTP/2 ei ollut testien aikaan tarpeeksi kehittynyt, käytettiin testeissä SPDY:ä sen sijaan. SPDY ja HTTP/2 ovat hyvin samankaltaisia kanavoinnin ja palvelimen työnnön kanssa.

Verkkoasiakas sijaitsee asiakaskoneella ja tekee sen läpi HTTP-pyyntöjä palvelimelle. Verkkoasiakas käyttää Google Chrome -selainta. Verkkosivulle on integroitu videotoistin. Perustoiminnoltaan se alkaa toistaa videota saatuaan viidennen osan videosta. Kaistaleveyden arvioinnissa se käyttää pessimististä arviointia, eli kerrointa 0,9. Bittinopeuden arviointiin käytetään painoja yhdistämään uusi arvio. Painot ovat 0,45 ja 0,55. Koodia päivitettiin käyttämään WS lähettämään tietoa välimuistin ja videon toiston tilasta. Käyttölogiikka on palvelimella, kun palvelin työntää sisältöä. Kaistanleveyden arviointi lasketaan WS-viestien perusteella ja lähetetään palvelimelle.

Testivideon palat ovat kaksi sekuntia pitkiä. Resoluutio on 1920x1080. Bittinopeus vaihtelee 2500 Kbps ja 8000 Kbps välillä. Asiakas- ja palvelinkoneet käyttävät Ubuntu 12.04 -versioita normaaleilla arvoilla. Verkkoyhteyden kaistanleveys on 940 Mbps ja viive yksi millisekunti. Verkkoyhteyden hallintaan käytettiin Linuxin liikenteenohjausta.

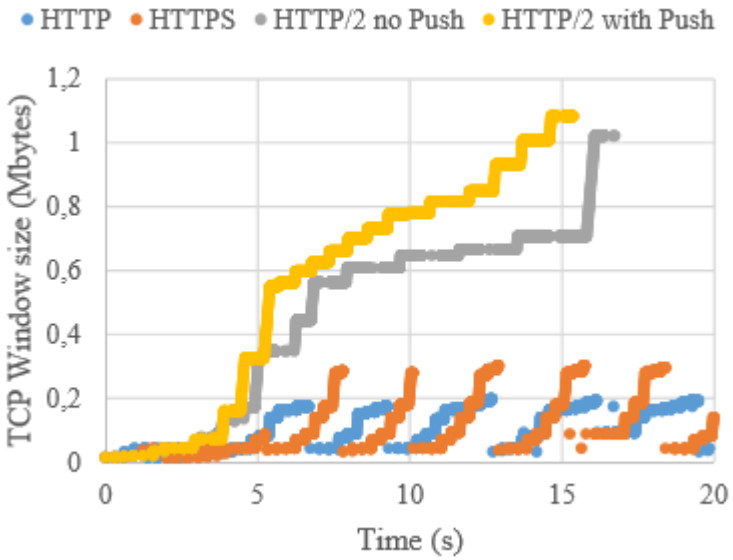
HTTP/1.1:n ja HTTP/2:n väliseen vertailuun kaistanleveys asetettiin 4,8 Mbps ja kiertoaika vaihtelemaan 50 ms ja 300 ms välille ilman pakettihäviötä. Eri asetuksilla ajettiin aina 10 testiä. Tulokset ovat kuvassa 4.8. Kuvassa HTTP/1.1 on esitetty sinisellä, suojattu versio punaisella. HTTP/2 ilman työntöä on taas esitetty harmaalla ja työnnön kanssa keltaisella. Kuvasta nähdään HTTP/2:n ja palvelimen työnnön tehokkuuden kasvavan kiertoajan kasvaessa. 50ms viiveen kanssa kaikki eri versiot ovat lähes tasoissa. 100ms kanssa HTTP/2 työnnön kanssa alkaa olla muita nopeampi. Työnnön kanssa kiertoajalla ei ole varsinaisesti enää väliä, sillä puskuria täytetään koko ajan. HTTP/2 työnnön kanssa onkin lähes yhtä nopea 50 ms ja 300 ms viiveiden kanssa, kun muut hidastuvat selvästi. 300 ms kiertoajalla HTTP/2 on 30% nopeampi kuin pelkkä HTTP/2 ja 115% nopeampi kuin HTTP/1.1.



Kuva 4.8 Videon latausajat kiertoajan kasvaessa. (Cherif et al. 2015)

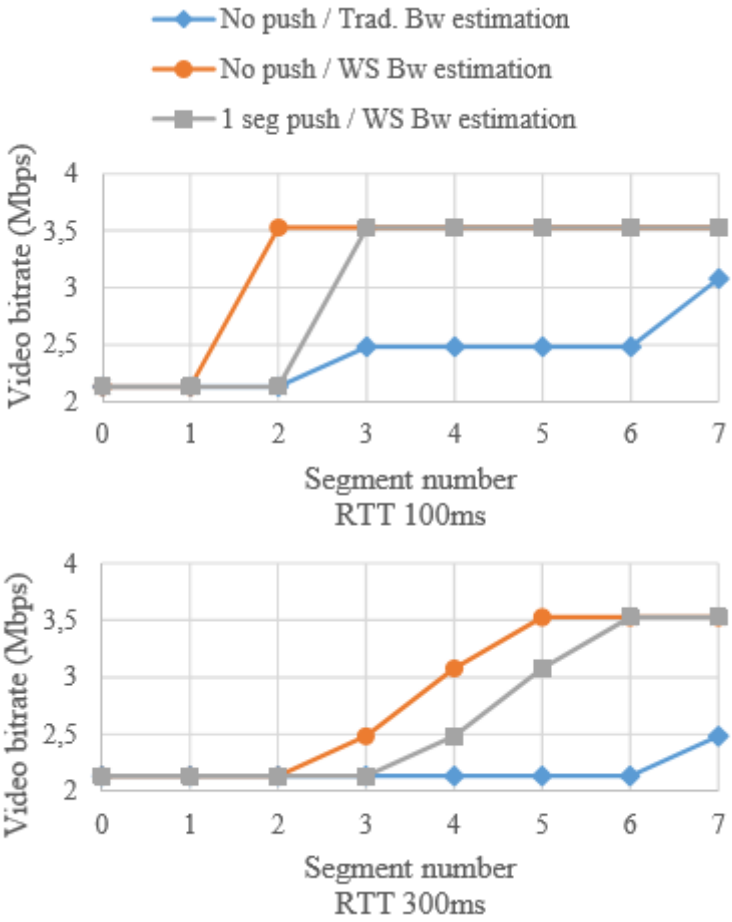
HTTP/2:n tehokkuuden lisäys yhdistetään TCP-vastaanottoikkunan koon kasvattamiseen. Vastaanottoikkuna tunnistaa maksimaalisen määrän dataa, jonka lähettäjä voi lähettää ilman TCP-vastausta vastaanottajalta. Ikkuna alkaa pienestä alkukoosta ja kasvaa jokaisen TCP-vastauksen mukaan, kunnes kynnysarvon jälkeen se ei enää kasva. Seurauksena suoritusteho rajoittuu ikkunankoon ja sen kasvun mukaan.

TCP-vastaanottoikkunaa testattiin 4,8 Mbps koon ja 300 ms kiertoajan kanssa. Tulokset löytyvät kuvasta 4.9. HTTP/1.1-testit sinisellä ja punaisella käyttävät selaimen sallimaa kuutta TCP-yhteyttä. HTTP/2-testit käyttävät yhtä TCP-yhteyttä. Testien mukaan resurssien työntäminen kasvattaa TCP:n vastaanottoikkunaa nopeammin, kuin normaalit HTTP-pyynnöt ja vastaukset. Näin ilman pyyntöjä välissä saavutetaan TCP-yhteyden maksimaalinen suoritusteho. Kuvasta 4.9 nähdään selvästi miten HTTP/1.1:n kanssa TCP-ikkuna sulkeutuu välillä ja maksimikoko onkin 0,3 MB tasolla. HTTP/2:n kanssa ikkuna ei sulkeudu ja se kasvaakin yli 1 MB kokoiseksi.



Kuva 4.9 TCP-vastaanottoikkunan kasvattaminen. (Cherif et al. 2015)

Samoilla asetuksilla (4,8 Mbps ja 300 ms) testattiin myös kaistanleveyden arviointia WS:n kanssa ja ilman. Testi löytyy kuvasta 4.10. Testeissä havaittiin, että sinisellä merkitty perinteinen tapa arvioida kaistanleveyttä kasvattaa bittinopeutta vasta seitsemännen video-osan kohdilla. Punaisella merkitty WS:n arviointikyky on niin tehokas, että bittinopeus kasvaa jo toisen video-osan kohdalla. Harmaalla merkityssä palvelimen työnnössä WS:n arviointi saadaan asiakkaalta vasta yhden osan viiveellä, joten se nostaa nopeutta vasta kolmannen osan kohdalla. WS:n käyttö HTTP/2-yhteyden sisällä mahdollistaa bittinopeuden kasvattamisen huomattavasti nopeammin ja lisätutkimusta WS:n laajentamiseen videoiden suoratoistoon pitäisi harkita.

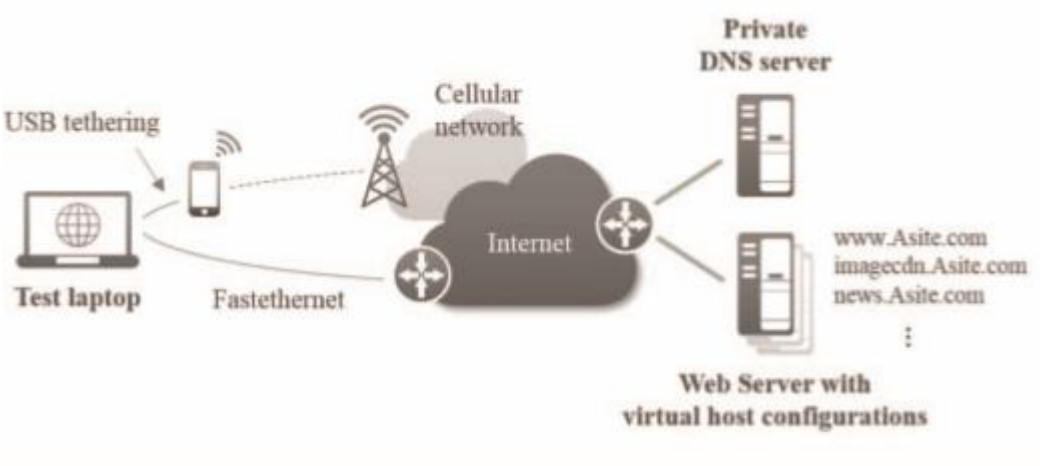


Kuva 4.10 DASH kaistanleveyden käyttö 100ms ja 300ms kiertoajoilla. (Cherif et al. 2015)

4.3 HTTP/2:n vaikutus usean domainin sivustoihin

Kim et al. (2015) tutki HTTP/2:n vaikutusta verkkoselauksen nopeuteen. Tarkoituksena on tutkia tehokkuutta tyypillisillä Koreassa sijaitsevilla sivustoilla, jotka käyttävät useaa domainia.

Tähän tarkoitukseen rakennettiin testialusta, jonka verkkoselain tukee HTTP/2:ta ja emuloi eri domaineita. Testialusta löytyy kuvasta 4.11. Tutkimuksessa käytettiin omaa DNS-palvelinta, joka vastaa nimikyselyistä testiympäristössä. Useamman verkkoympäristön simuloimiseksi käytettiin myös älypuhelinta mahdollistamaan tietokoneen pääsy matkapuhelinverkkoon.



Kuva 4.11 Testialustan kokoonpano. (Kim et al. 2015)

Useita verkkosivustoja Alexan 15. parhaan joukosta Koreassa haettiin ja kopioitiin, jonka jälkeen tehokkuusarvioinnit suoritettiin eri verkko-olosuhteissa. Näistä verkkosivuistoista kuusi valittiin edustamaan eri kategorioita: portaalia, uutisia ja ostossivustoja. Taulukosta 4.1 löytyy sivustojen eri tiedot matkapuhelinverkon testeihin: eri resurssien lukumäärä, sivun koko kilotavuina ja domainien määrä. Domainien määrä johtuu käytetystä palastelusta nopeuttamaan sivustojen toimintaa. Erilaisia sivustoja valittiin selvittämään eroja erikokoisista sivustoista ja eri domainien määrillä. Samat sivustot tutkittiin myös tietokoneella, mutta sivustojen tiedot muuttuvat hieman. Tietokoneella tutkitut tiedot löytyvät taulukosta 4.12.

Taulukko 4.1 Matkapuhelinverkossa testatut sivustot ja niiden tiedot. (Kim et al. 2015)

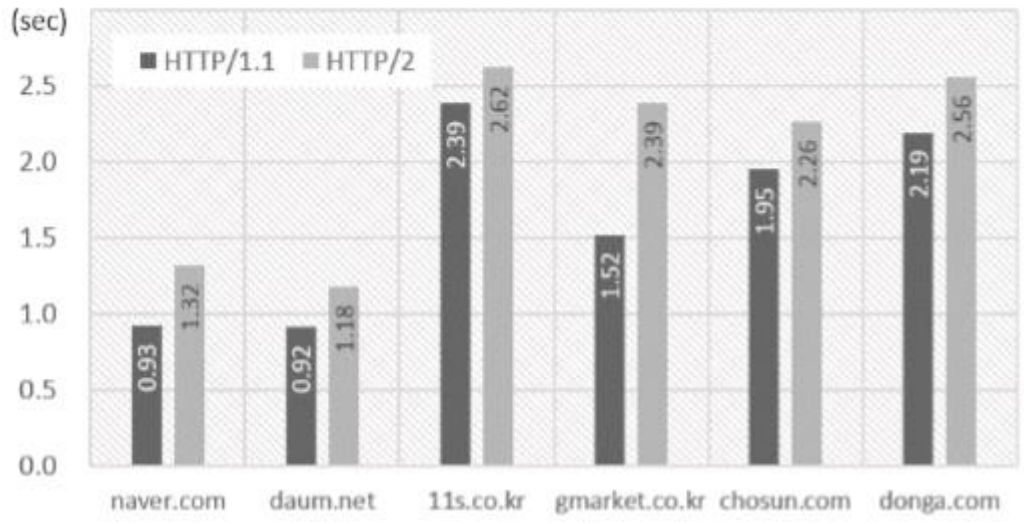
	Test pages	Number of resources	Size of page (KB)	Number of domains
Portal	m.naver.com	68	1737	12
	m.daum.net	48	1021	4
Shopping	m.11st.co.kr	130	5937	5
	m.gmarket.co.kr	69	2645	5
News	m.chosun.com	120	2354	18
	m.donga.com	106	1824	12

Taulukko 4.2 Tietokoneella testatut sivustot ja niiden tiedot. (Kim et al. 2015)

	Test pages	Number of resources	Size of page (KB)	Number of domains
Portal	www.naver.com	90	2068	14
	www.daum.net	74	1298	13
Shopping	www.11st.co.kr	140	2809	4
	www.gmarket.co.kr	174	2216	9
News	www.chosun.com	184	2559	7
	www.donga.com	276	8173	14

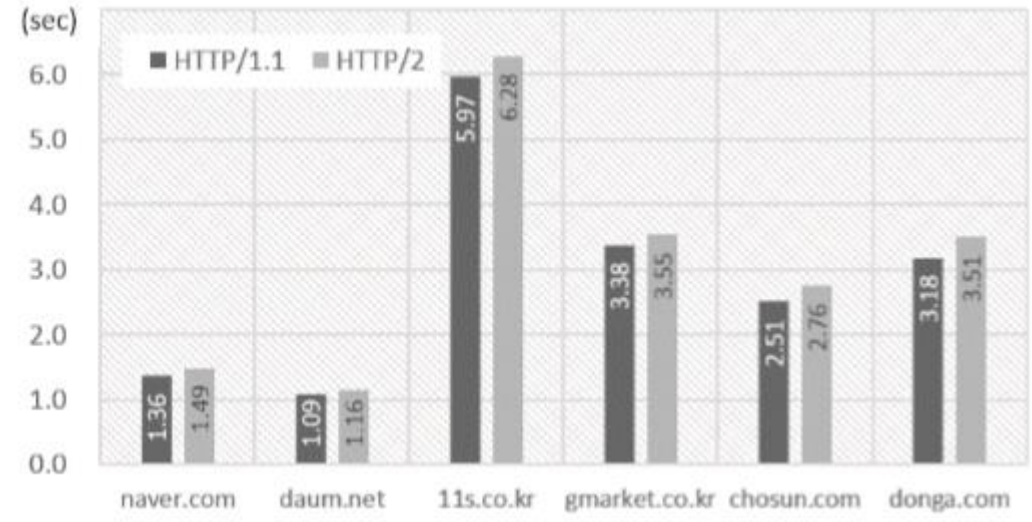
Mittauksissa tutkittiin sivujen kaikkien resurssien latausaikaa. Kaikki testit ajettiin kymmenen kertaa ja selaimen välimuisti tyhjennettiin latausten välillä.

Ensimmäiseksi testit ajettiin LTE-matkapuhelinverkossa. Keskiarvoinen kaistanleveys testeissä oli 114.3 Mbps ja viive 25 ms. Testitulokset löytyvät kuvasta 4.14. Näistä testeistä havaittiin HTTP/2:n kasvattavan latausaikaa 28 prosenttia. Yksinkertaisempien sivujen kanssa latausajat huonontuivat enemmän.



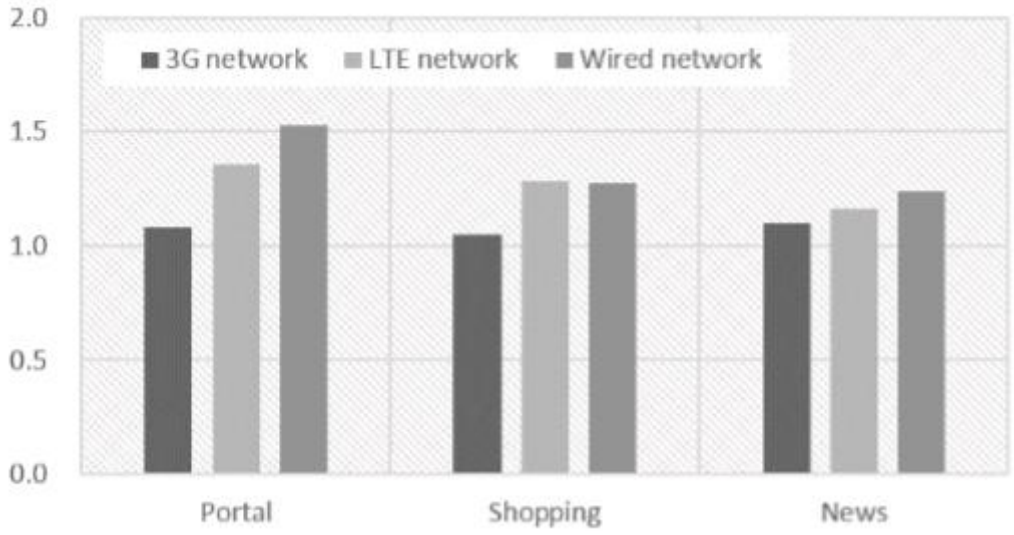
Kuva 4.14 LTE-matkapuhelinverkossa saadut sivustojen latausajat. (Kim et al. 2015)

Samat testit ajettiin 3G-matkapuhelinverkossa. Keskiarvoinen kaistanleveys testeissä oli 8 Mbps ja viive 34 ms. Testitulokset löytyvät kuvasta 4.15. Pienemmillä nopeuksilla HTTP/2 on edelleen 8 % hitaampi kuin HTTP/1.1. Matkapuhelinverkon tuloksista pääteltiin HTTP/2:n sopivan hitaammille verkoille ja sivuille, jotka käyttävät vähemmän domaineja.



Kuva 4.15 3G-matkapuhelinverkossa saadut sivustojen latausajat. (Kim et al. 2015)

Lopuksi testit ajettiin myös tietokoneella. Keskiarvoinen kaistanleveys testeissä oli 93,4 Mbps ja viive 7 ms. Testitulokset kuvastivat samaa toimintaa kuin matkapuhelinverkossa. Kuvasta 4.16 löytyvät kaikki mittaustulokset ilmaistuna suhteella. Yli yhden olevissa tuloksissa HTTP/2 on hitaampi kuin HTTP/1.1.



Kuva 4.16 Latausaikojen erot eri verkoilla kuvattuna suhteella. Yli yhden olevissa tuloksissa HTTP/2 on hitaampi kuin HTTP/1. (Kim et al. 2015)

Tämä tutkimus näyttääkin siltä, että HTTP/2 ei anna etua erittäin pienen viiveen verkoissa. 50 ms viiveen kanssa muissa testeissä HTTP/2 on ollut yhtä nopea tai nopeampi kuin HTTP/1.1. Testi herättääkin huomattavia kysymyksiä protokollan tehokkuudesta.

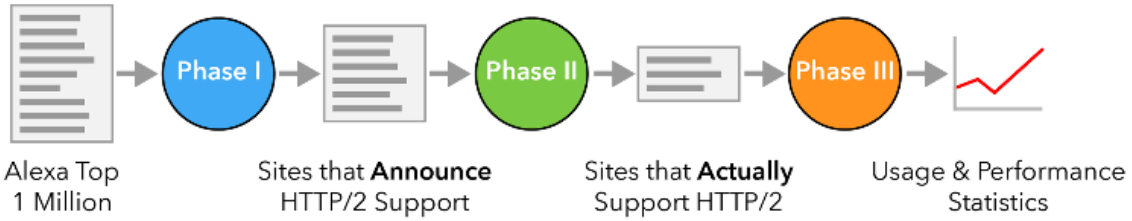
4.4 HTTP/2:n käyttö

Varvello et al. (2015) julkaisi tutkimuksen HTTP/2:n käyttöasteesta ja toiminnasta. Heidän tutkimuksistaan syntyi myös verkkosivu HTTP/2 Dashboard (http://isthewebhttp2yet.com/). Tälle verkkosivulle he jatkavat tutkimuksensa tulosten julkaisua.

HTTP/2 Dashboard tutkii HTTP/2-tuen levinneisyyttä Internetissä. Heidän tutkimuksensa toimii kolmessa vaiheessa, joka on kuvattu myös kuvassa 4.17:

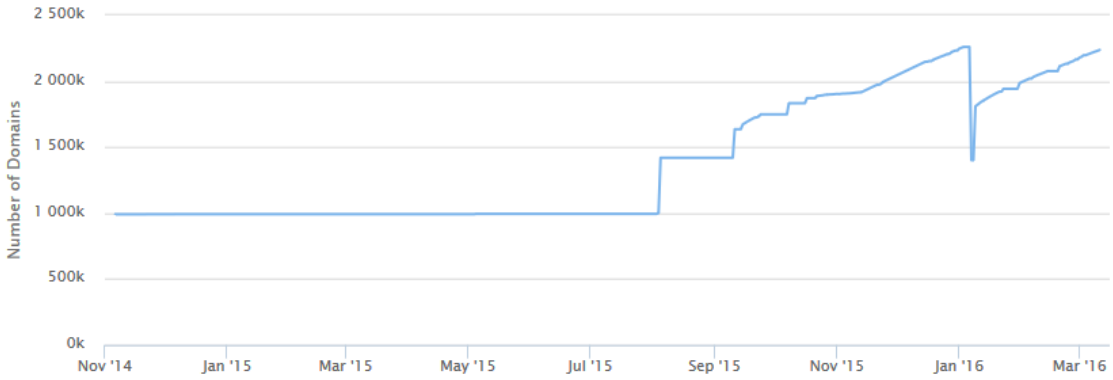
- Ensimmäisessä vaiheessa tutkitaan päivittäin miljoona ensimmäistä sivustoa Alexan listasta. Tämä tehdään käyttämällä TLS:n laajennuksia NPN ja ALPN.
- Toisessa vaiheessa tutkimuspisteet Yhdysvalloissa ja Espanjassa pyrkivät hakemaan juuriobjektin sivuilta, jotka ilmoittavat tukevansa HTTP/2:ta. Tästä saadaan lista sivuista, jotka osittain ja kokonaan tukevat HTTP/2:ta. Osittain tukeviksi sivuiksi lasketaan kaikki ne sivut, jotka vastaavat jollain tavoin käyttäen HTTP/2:ta, vaikka se olisi vain virheilmoitus.

- Kolmannessa vaiheessa noudetaan viikoittain jokainen täysin tukeva sivu, käyttäen HTTP/2- ja HTTP/1.1-protokollia. Tästä talletetaan tieto toiminnasta kuten sivun latausaika ja käytettyjen TCP-yhteyksien määrä.



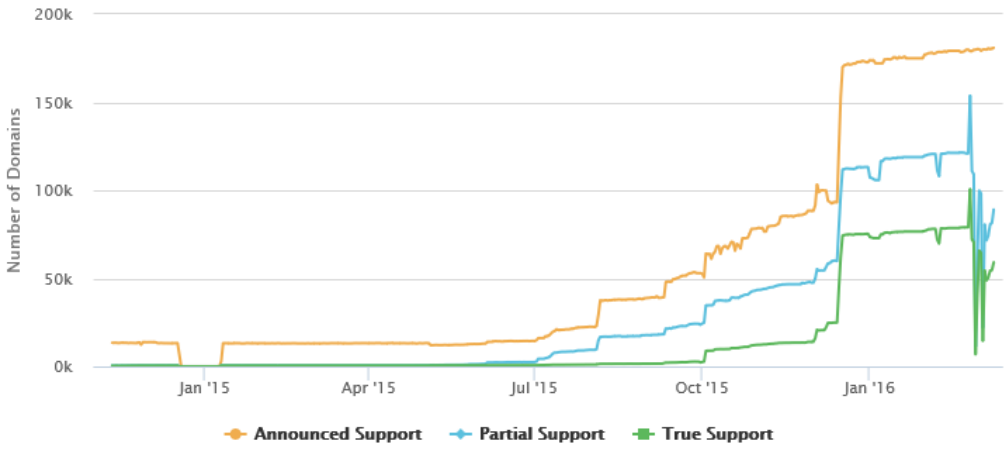
Kuva 4.17 HTTP/2 Dashboardin kolmivaiheinen testausprosessi. (Varvello et al. 2016)

Tutkittavien sivustojen määrä pysyi aluksi samana, koska käytettiin alkuperäistä Alexan listaa. Syyskuun 2015 alusta alkaen lista päivitetään joka päivä lisäämällä uudet sivut Alexan miljoonan sivun joukosta. Näin tutkittavien sivustojen määrä kasvaa koko ajan. Tutkittavien sivustojen määrä löytyy kuvasta 4.18.



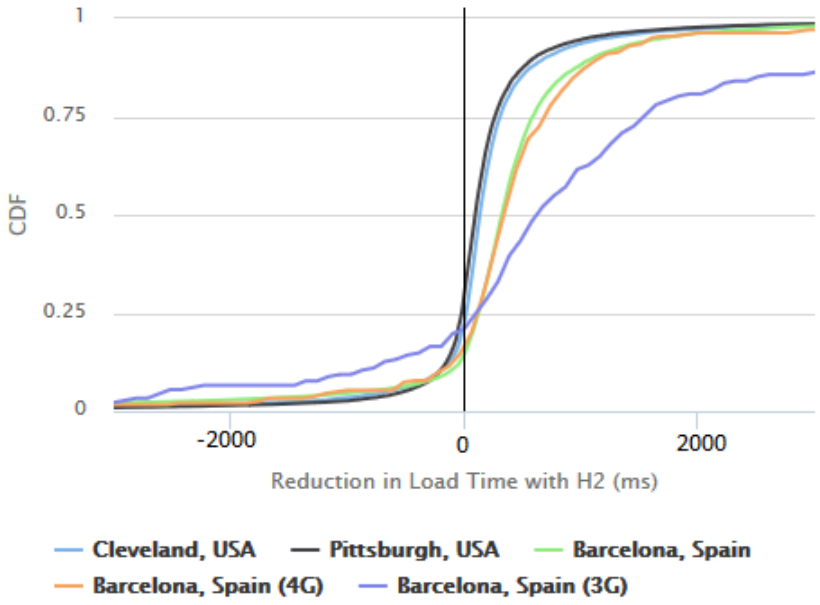
Kuva 4.18 HTTP/2 Dashboardin tutkittavien sivustojen kehitys yli kahteen miljoonaan sivustoon. (Varvello et al. 2016)

Maaliskuussa 2016 HTTP/2:ta tukevien sivustojen määrä ja kehitys on esitetty kuvassa 4.19. Oranssilla löytyy 180 194 sivua, jotka ilmoittavat tukevansa HTTP/2:ta. Sinisellä on 89 149 sivua, jotka osittain tukevat HTTP/2:ta. Vihreällä ovat ne 59 114 sivua, joista löytyy täysi tuki. Täyden tuen sivujen määrä on kasvanut toukokuusta 2015 huomattavasti, sillä silloin täyttä tukea tarjosi vain 600 sivua. Nämä olivat lähinnä Googlen ja Twitterin sivuja. Tutkittujen sivujen määrä on tippunut vuoden 2016 alussa tuntemattomasta syystä hetkellisesti ja se näkyy myös selvänä piikkinä tuettujen sivujen kuvaajassa.



Kuva 4.19 HTTP/2 tuettujen sivujen määrä kaikista tutkimuksessa mukana olevasta sivustosta. (Varvello et al. 2016)

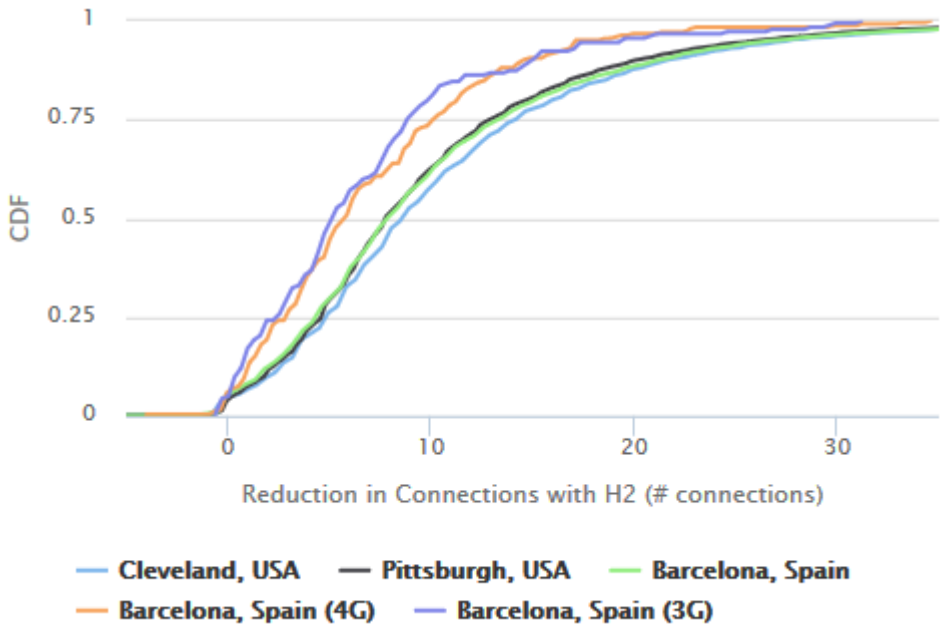
Tulokset on esitelty kumulatiivisina jakaumafunktiona (CDF). CDF kuvaa mittausten sarjaa kertomalla todennäköisyyden valittua mittaussuuretta kohtaan. Seuraavaksi esitellään mittaustulokset. Ensimmäisenä esitetään sivujen latausaikoja kuvassa 4.20 eri mittauspisteissä. Kuvan mittaustulokset ovat HTTP/2:n latausaikojen lasku HTTP/1.1 vastaan. Vaaleansinisellä ja mustalla ovat mittauspisteet Yhdysvalloissa ja vihreällä Espanjassa. Lisäksi keltaisella ja tumman sinisellä ovat 4G- ja 3G-mittaukset Espanjasta. Jakajan vasemmalla puolella oleva määrä on hitaampia HTTP/2 kanssa. Jakajan oikealla puolella olevat sivustot ovat nopeampi HTTP/2 kanssa. Kuvasta havaitaan, että 15-30 prosenttia sivuista on hitaampia HTTP/2:n kanssa riippuen mittauspisteestä. Vastaavasti 70-85 prosenttia sivuista on nopeampia.



Kuva 4.20 Sivujen latausaikojen lasku HTTP/2:n kanssa useista eri mittauspisteistä. (Varvello et al. 2016)

Erityistä huomiota kannattaa kiinnittää mobiilin nopeutuneisiin latausaikoihin. Aikaisemmat tutkimukset mobiilista SPDY:n kanssa ovat osoittaneet huonompaa toimintakykyä. Tutkimuksessa oleva ero voi selittyä sillä, että käytetään todellisia sivustoja ja toimintatapoja. TCP-yhteyksiä avataan tarvittava määrä eri domaineihin, eikä pakoteta toimintaa yhteen yhteyteen. Yhden yhteyden kanssa HTTP/2 on herkkä pakettien menetykselle, mutta domainin palastelu auttaa tässä ongelmassa.

Käytettävien yhteyksien määrä ei voi laskea yhteen edes HTTP/2:den kanssa, koska sivustot jakavat tietonsa useaan domainiin ja jokaiseen joudutaan ottamaan TCP-yhteys. Huomattava yhteysmäärien väheneminen löytyy kuitenkin kuvasta 4.21. Värit ovat samat kuin edellisessä kuvassa.



Kuva 4.21 TCP-yhteyksien määrän lasku käytettäessä HTTP/2:ta. (Varvello et al. 2016)

4.5 Johtopäätökset tutkimustuloksista

Ensimmäisessä testissä nähtiin HTTP/1.1:n latausaikojen kasvavan suhteessa pyyntöjen määrään. Testit ajettiin yksinkertaistetussa ympäristössä, jotta pelkkä protokollien ero selviäisi. HTTP/2 osoitti hyvää suorituskykyä ja varsinkin viiveen kasvaessa huomattavasti parempaa kuin HTTP/1.1. Mobiilitesteiksi tarkoitetut kuuden prosentin pakettihäviötestit vastaavasti osoittivat huonoa suorituskykyä HTTP/2:n kanssa.

Toisessa testissä tutkittiin videon suoratoiston aloittamista eri HTTP-versioilla. Palvelimen työnnön kanssa TCP-yhteys kasvaa nopeasti maksimaaliseen suorituskykyyn. Verkonkiertoajan kasvaessa HTTP/2:n kanssa videon latausaika voi parantua jopa 50 pro-

senttia. Yhteysnopeuden kasvattamiseen testissä esiteltiin WebSocket-ohjausdata toimimaan HTTP/2-yhteyden sisällä. Tämä ohjausdata kasvatti TCP-yhteyden nopeasti maksimitasolleen.

Kolmannessa testissä tutkittiin myös resurssien latausaikaa. Käytössä oli erittäin laajan kaista ja erittäin pieni viive, joka vaihteli 7-34 millisekunnin välillä. Kaikissa testeissä saatiin HTTP/2 osoittamaan huonompaa suorituskykyä kuin HTTP/1.1.

Neljäs testi tehtiin oikeille palvelimille ja varsinkin pakettihäviön vaikutus mobiililaitteilla oli huomattavan pientä. Mobiililaitteiden latausnopeus oli myös aikaisempia ennusteita vastaan nopeampaa HTTP/2 kanssa, koska useampaan domainiin yhteyden muodostaminen pienensi ensimmäisessä testissä havaittuja pakettihäviön ongelmia. Aikaisemmissa testeissä testiympäristöt pakottivat mobiilidatan yhteen TCP-yhteyteen.

Huomattavaa on, että kolme ensimmäistä testiä ovat alustavia tutkimuksia. Kaikki painottavat, että todelliseen HTTP/2:n suorituskyvyn arviointiin tarvitaan lisää testejä. Neljännessä tutkimuksessa päästiin jo oikeisiin Internetin yli oleviin testeihin oikeille palvelimille. Vaikka ei voida väittää sen olevan täydellinen osoitus HTTP/2:n suorituskyvystä, niin ainakin se on lähinnä sitä. HTTP/2 osoittaakin huomattavaa nopeusetua aikaisempaan HTTP/1.1:een nähden suurimmassa osassa sivustoja. Nopeusetua voidaan luultavasti kasvattaa optimoimalla sivustoja uudelle protokollalle palvelimien päässä.

Neljäs Varvellon ja kumppanien tutkimus on selvästi laajin ja saa suurimman luottamukseni. Sen pohjalta voidaan sanoa HTTP/2:n nopeuttavan Internetliikennettä 70-85% tapauksissa. Tutkimus osoittaa myös alkuvaiheen, jossa HTTP/2-tuki vielä on. Tuen laajenemista koskemaan suurempaa osaa Internetistä täytyy vielä odottaa. Tutkimus osoittaa kuitenkin tuen olevan leviämässä.

Palvelimen työntö osoittautui jo näissä alustavissa testeissä mahdollisiksi verkkoliikenteen nopeuttajaksi. Palvelimen työntöä ei hyödynnetä vielä juurikaan millään verkkosivulla. Olemassa olevat palvelimet ja sovellukset täytyy päivittää tukemaan HTTP/2:ta ja tutkia parhaita keinoja nopeuttamaan omaa toimintaansa. Kaikkien palvelimien saaminen tukemaan HTTP/2:ta tulee olemaan jo suuri ponnistus Internetin nopeuttamiselle.

Työssä esiteltiin neljä tutkimusta HTTP/2:n tehokkuudesta eri tilanteissa. Ensimmäiset testit tehtiin suojaamattomina, jottei TLS-suojaus tai muut ulkoiset tekijät vaikuta tuloksiin omissa testiympäristöissään. Yhteinen tekijä testeissä on, että HTTP/2 näyttää kestävän verkon kiertoajan kasvamista erityisen hyvin ja tarjoavan silloin huomattavaa parannusta. Yhden TCP-yhteyden käyttäminen tekee kuitenkin siitä haavoittuvaisen pakettihäviölle. Häviö koskee kaikkia voita samanaikaisesti ja laskee suoritustehoa huomattavasti. Aikaisemmat SPDY:llä tehdyt testit olivat jo osoittaneet tämän ja se sai vahvistusta ensimmäisestä de Saxcé et al. tutkimuksesta. Tämä ongelma poistuu kuitenkin jo käytössä olevalla domainin palastelulla, kuten neljännestä tutkimuksesta havaittiin. Oikeat olosuhteet eivät pakota HTTP/2:ta yhteen yhteyteen, kuten testitilanteissa oli pakotettu.

HTTP/2-protokolla nopeuttaa jo itsellään verkkosivujen latausaikoja ja vähentää viivettä. Jatkuva HTTP/2:n leviäminen useammille ja useammille verkkosivuille tulee vaikuttamaan palvelinpään optimoinnin kehittymiseen myös uudelle protokollalle.

5. YHTEENVETO

Tässä työssä käytiin läpi HTTP-protokollaan tulleet muutokset, sekä esiteltiin alla olevan TCP-protokollan toimintaa ja HTTP:n historiaa. Erityistä huomiota käytettiin kuvaamaan HTTP/2-protokollan rakenne. Työssä esiteltiin neljä tutkimusta HTTP/2:n suorituskyvystä aikaisempaa HTTP/1.1:tä vastaan.

HTTP/2:ta ja sen vaikutuksia tutkittiin erityisesti viiveen näkökulmasta. Viive on hallitseva tekijä verkkoselauksessa yhdessä kaistanleveyden kanssa. Viive on kuitenkin muodostunut viimeisien vuosien aikana verkkoselauksen pullonkaulaksi. Kaistanleveyden kasvattamisella ei pystytä enää saavuttamaan merkittävää suorituskyvyn kasvua. Viiveen pienentäminen kasvattaa suorituskykyä aina merkittävästi. Tämä Mike Belshen tutkimuksen tulos johti SPDY-protokollan kehittämisen alkuun. SPDY-protokolla kehittyi sitten HTTP/2-protokollaksi.

HTTP toimii kuljetuskerroksen protokolla TCP:n päällä. TCP:n ominaisuudet liittyvät vahvasti HTTP:n toimintaan ja siksi myös TCP-protokolla esitellään työssä. TCP on yhteyspohjainen ja on erittäin hyvä siirtämään myös suuria määriä dataa. Muodostetun yhteyden kyky siirtää dataa kasvaa nopeasti, mutta HTTP:n aikaisemmat versiot avaavat ja sulkevat TCP-yhteyksiä ilman, että tätä suorituskykyä pystyttiin täysin hyödyntämään. TCP tarjoaa luotettavan tiedonvälityksen sen päällä toimiville protokollille.

Työssä käytiin läpi HTTP:n historiaa ja kehitystä nykyiseen muotoonsa; kuinka yksinkertaisesta yksirivisestä käskystä kehittyi HTTP/2-protokolla. Tässä käytiin läpi myös yleisimmin käytössä olevia kiertoteitä, joita palvelimilla käytetään nopeuttamaan aikaisempien HTTP:n versioiden toimintaa. HTTP/2 pyrkii siirtämään näitä toiminnollisuuksia itse protokollaan. Osaa ei kuitenkaan voida korvata protokollalla ja esimerkiksi domainin palastelu vaikuttaa sellaiselta. Varsinkin 3G- ja 4G-yhteyksillä palastelu parantaa HTTP/2:n toimintaa vähentämällä pakettihäviön vaikutusta. Tässä vaiheessa tutustuttiin myös HTTP:n ongelmiin ja siihen, miksi uutta protokollaan tarvitaan. HTTP/2:n pohjana oleva SPDY-protokolla esitettiin myös antamaan kuva kehityksestä, yhdenkaltaisuuksista ja eroista uuteen protokollaan.

HTTP/2-protokollan rakenteeseen ja toimintaan pureuduttiin tarkasti, mutta pyrkien välttämään liian yksityiskohtaista kuvausta. Protokollasta oli tarkoitus antaa huolellinen yleiskuva. Kaikki HTTP/2-protokollan osat käytiin läpi. HTTP/2 on binäärinen protokolla ja näin huomattavasti helpompi jäsentää, ja se mahdollistaa pyyntöjen kanavoinnin. Näin HTTP/2 pystyy lähettämään yhden yhteyden sisällä pyyntö- ja vastausviestit. Tähän HTTP/2 esittelee vuot, jotka ovat itsenäisiä ja kahdensuuntaisia ketjuja kehyksiä asiakkaan ja palvelimen välillä. Yhden yhteyden sisällä voi olla useita voita. Vuot kanavoidaan yhteen yhteyteen ja tunnisteiden avulla tiedot pysyvät oikeassa vuossa.

Työssä käytiin läpi protokollan tarjoamia toiminta vuonhallintaan ja tutustuttiin vuon priorisointiin. Priorisoinnissa vuolle voidaan antaa tärkeyttä kuvaava arvo, jolloin eri voita pystytään käsittelemään tärkeysjärjestyksessä. Uutena ominaisuutena on myös palvelimen työntö, jossa palvelin voi lähettää useita vastauksia yhteen pyyntöön. Näin palvelin voi ennakoida tulevia pyyntöjä ja poistaa kiertoaikoja liikenteestä ja näin vähentää viivettä. Aiemmissa HTTP:n versiossa otsakkeita ei pakattu, mutta HTTP/2 pakkaa otsakkeensa. Pakkaus tapahtuu käyttämällä staattista ja dynaamista taulukkoa toiminnoista, jolloin toiminnot voidaan ensimmäisen kerran jälkeen lähettää vain numeroina koko pyynnön sijasta. Tämä pienentää otsakkeiden kokoa.

HTTP/2:n kuvauksessa käytiin läpi myös yhteydenmuodostus uuden protokollan kanssa. Yleisimmin tämä tapahtuu päivittämällä HTTP/1.1-yhteys HTTP/2-yhteydeksi, mutta myös ennakkotiedolla uudemman protokollan tuesta yhteys voidaan avata suoraan. HTTP/2:n kehys ja kehysten eri tyypit esitettiin. Tutustuttiin nopeasti eri virhetyyppeihin ja protokollan turvallisuuteen eri hyökkäyksiä vastaan. Protokolla pyrkii suojaamaan toimintaansa antamalla virheitä, jotka estävät hyökkäyksiä sitä vastaan.

Seuraavaksi esiteltiin neljä tutkimusta HTTP/2:n toiminnasta. Toimintaa verrattiin edellistä HTTP/1.1-versiota vastaan. Kolme ensimmäistä tutkimusta olivat alustavia ja omassa testiolosuhteissaan rajoittunutta testiä. Viimeinen Varvello et al. tutkimus oli selvästi laajin. Tutkimus on jatkuva ja kohdentuu miljooniin tutkittaviin sivuihin, joista selvitetään tuki HTTP/2:lle ja tuettujen sivujen toimintaa verrataan ladattuna molemmilla HTTP:n versioilla.

Ensimmäinen de Saxce et al. tutkimus osoittaa HTTP/2:n suorituskyvyn pysyvän tasaisena riippumatta pyyntöjen määrästä, jos datan määrä ei kasva. Tässä nähdään selvä ero HTTP/1.1:een, sillä pyyntöjen määrän kasvu hidastaa sen toimintaa selvästi. Muutenkin tutkimus osoittaa HTTP/2:n reagoivan erittäin hyvin viiveeseen ja suorituskyvyn olevan selvästi HTTP/1.1:tä parempi varsinkin kiertoajan kasvaessa. 200 ms viiveillä HTTP/1:n keskiarvoinen viive oli 9.0 s ja HTTP/2:n 4.2s.

Toinen Cherif et al. tutkimus keskittyi videon suoratoistoon ja sen aloittamiseen nopeammin. Tässäkin HTTP/2 osoitti hyvää suorituskykyä kiertoajan kasvaessa. Tutkimuksessa esiteltiin myös palvelimen työntöä käytännössä ja sen mahdollistamaa etua. Tässäkin tapauksessa etu kasvoi huomattavaksi verkon kiertoajan kasvaessa.

Kim et al. tutkimus osoittaa HTTP/2:n olevan erittäin laajalla kaistalla ja pienellä viiveellä olevan hitaampi kuin HTTP/1.1.

Neljäs Varvello et al. tutkimus onkin paljastavin. Sen laajuus ja monien mittauspaikkojen käyttö oikeille palvelimille saa luottamukseni. Sen pohjalta voidaan todeta HTTP/2:n nopeuttavan 70-85 prosenttia verkkosivustoista riippuen sijainnista ja käytetystä yhteydestä. Tutkimus myös paljastaa HTTP/2-tuen olevan leviämässä. Aikaisemmissa tutkimuksissa pelätty hitaampi toiminta 3G- ja 4G verkkojen yli osoittautuu myös turhaksi.

HTTP/2-protokolla nopeuttaa jo itsellään verkkosivujen latausaikoja ja vähentää viivettä. Jatkuva HTTP/2:n leviäminen yhä useammille ja useammille verkkosivuille tulee vaikuttamaan palvelinpään optimoinnin kehittymiseen myös uudelle protokollalle.

LÄHTEET

Adi, E., Baig, Z., Peng Lam, C., Hingston, P. (2015). Low-Rate Denial-of-Service Attacks against HTTP/2 Services. IT Convergence and Security (ICITCS), 2015 5th International Conference on. pp. 1-5.

Allison, C., Bakri, H. (2015). How the Web was Won: Keeping the computer networking curriculum current with HTTP/2. Frontiers in Education Conference (FIE). pp.1-9.

Belshe, M. (2010). More bandwidth doesn't matter much. Saatavissa (viitattu 9.3.2016): https://www.belshe.com/2010/05/24/more-bandwidth-doesnt-matter-much/

Belshe, M., Peon, R. & Thomson M. (2015). Hypertext Transfer Protocol Version 2. RFC 7540. Internet Engineering Task Force. Saatavissa (viitattu 7.12.2015): https://tools.ietf.org/html/rfc7540

Cherif, W., Fablet, Y., Nassor, E., Taquet, J. & Fujimori, Y. (2015). DASH fast start using HTTP/2. Proceedings of the 25th ACM Workshop on Network and Operating Systems Support for Digital Audio and Video. pp.25-30.

Dordal, P. (2014). An Introduction to Computer Networks. Department of Computer Science Loyola University Chicago. Saatavissa (viitattu: 7.3.2016): http://intronetworks.cs.luc.edu/

Fielding, R. & Reschke, J. (2014). Hypertext Transfer Protocol (HTTP/1.1). RFC 7230. Internet Engineering Task Force. Saatavissa (viitattu 27.2.2016): https://tools.ietf.org/html/rfc7230

Google (2015). SPDY: An experimental protocol for a faster web, verkkosivu. Saatavissa (viitattu 20.12.2015): http://www.chromium.org/spdy/spdy-whitepaper

Grigorik, I. (2013a). Making the Web Faster with HTTP 2.0. Communications of the ACM. Vol.56(12), pp.42-49.

Grigorik, I. (2013b), High Performance Browser Networking. O'Reilly Media. 361 p.

Grover, S., Kang, X., Kounavis, M., Berry, F. (2009). https://everywhere! Encrypting the Internet. Intel Technology Journal. Vol.13(2). pp.66-79.

Hodson R. (2015). HTTP/2 For Web Developers, verkkosivu. Saatavissa (viitattu 7.3.2016): https://blog.cloudflare.com/http-2-for-web-developers/

Httparchive. (2016). Interesting stats, verkkosivu. Saatavissa (viitattu 17.1.2016): http://httparchive.org/

Kim, H., Lee, J., Park, I., Kim, H. & Hur, D. (2015). The upcoming new standard HTTP/2 and its impact on multi-domain websites. Network Operations and Management Symposium. pp.530-533.

Krishnamurthy, B., Mogul, JC., Kristol, DM. (1999). Key differences between HTTP/1.0 and HTTP/1.1. Computer networks-the international journal of computer and telecommunications networking. Vol.31(11-16), pp.1737-1751.

Mueller, C., Lederer, S., Timmerer, C. & Hellwagner, H. (2015). Dynamic Adaptive Streaming Over HTTP/2.0. Multimedia and Expo (ICME). p.1-6.

Obren, M. & Howell, B. (2014). The tyranny of distance prevails: HTTP protocol latency and returns to fast fibre Internet access network deployment in remote economies. Annals of Regional Science. Vol.52(1), p.65-85.

Peon, R. & Ruellan, H. (2015). HPACK: Header compression for HTTP/2. RFC 7541. Internet Engineering Task Force. Saatavissa (viitattu 17.1.2016): https://tools.ietf.org/html/rfc7541

Sandvine Report. (2014). Encrypted Web Traffic More Than Doubles After NSA Revelations, verkkosivu. Saatavilla (viitattu 11.3.2016): http://www.wired.com/2014/05/sandvine-report/

de Saxce, H., Oprescu, I. & Yiping, C. (2015). Is HTTP/2 really faster than HTTP/1.1?. Computer Communications Workshops (INFOCOM WKSHPS). pp.293-299.

Stallings, W. (2000). Data & Computer Communications 6th edition. Prentice Hall. 810 p.

Stenberg, D. (2015). http2, background, the protocol, the implementations and the future. Saatavilla (viitattu 12.3.2016): https://daniel.haxx.se/http2/http2-v1.9.pdf

Tarreau, W., Jeffries, A., de Croy, A., Kamp, P-H. (2012) Proposal for a Network-Friendly HTTP Upgrade. Network Working Group. Saatavissa (viitattu 12.5.2016): https://tools.ietf.org/html/draft-tarreau-httpbis-network-friendly-00

Trace, R., Foresti, A., Singhal, S., Mazahir, O., Nielsen, H., Raymor, B., Rao, R., Montenegro, G. (2012). HTTP Speed+Mobility. Network Working Group. Saatavissa (viitattu 12.5.2016): https://tools.ietf.org/html/draft-montenegro-httpbis-speed-mobility-02

Torii, Y. (2016). Communications, Specification & Impelementation of HTTP/2 That Every Developer Has To Know. Research at Work Applications. Saatavissa (viitattu 14.1.2016): http://research.worksap.com/research/httpresearch-worksap-comresearchcommunication-specification-implementation/

Undertow. (2015). An in depth overview of HTTP/2, verkkosivu. Saatavilla (viitattu 10.3.2016): http://undertow.io/blog/

Varvello, M., Schomp, K., Naylor, D., Blackburn, J., Finamore, A., Papagiannaki, D. (2016). Monitoring the adoption and performance of HTTP/2 on the Web. Saatavissa (viitattu 11.3.2016): http://isthewebhttp2yet.com/

Varvello, M., Schomp, K., Naylor, D., Blackburn, J., Finamore, A., Papagiannaki, D. (2015). To HTTP/2, or Not To HTTP/2, That Is The Question. Saatavissa (viitattu 12.3.2016): http://arxiv.org/pdf/1507.06562.pdf

White, G., Mule, J-F., Rice, D. (2012). Analysis of SPDY and TCP Initcwnd. Network Working Group. Saatavissa (viitattu 26.2.2016): https://tools.ietf.org/html/draft-white-httpbis-spdy-analysis-00